Open Geospatial Consortium, Inc.

Date: 2010-08-02

Reference number of this document: OGC 10-060r1

Category: OGC Public Engineering Report

Editor: Johannes Echterhoff

OWS-7 Event Architecture Engineering Report

Warning

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

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

License Agreement

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

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

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

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

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

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

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

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

Preface

This document extends and revises OGC document 09-032, the "OGC® OWS-6 SWE Event Architecture Engineering Report". It incorporates the results of the OWS-7 Event Architecture cross thread activities.

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

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

Contents

Page

Figures

Page

Tables

Page

Listings

Page

OGC® OWS-7 Event Architecture Engineering Report

1 Introduction

1.1 Scope

This OGC™ document is applicable to use cases in which event-driven architecture principles are applied in Spatial Data Infrastructures.

The document specifies publish/subscribe functionality for OGC web services. This is done by first defining an abstract publish / subscribe model and then deriving functional requirements from this model.

Event channels – a concept to facilitate subscribing for and filtering of events – is developed and explained.

An evaluation of WS-Notification regarding the extent in which it realizes the publish / subscribe requirements is provided.

The document presents an approach for discovery of event service metadata. This allows clients to search for services that support certain events, event channels and filter functionality.

This document provides an overview of security threats that apply to the Event Architecture in general and an Event Service in particular. Ways to mitigate / prevent these threats using different security measures are discussed with respect to a WS-* / SOAP and RESTful environment.

Various aspects of filtering / processing events through an Event Service are discussed.

The document provides guidelines for inclusion of publish / subscribe functionality in other standards.

An overview of standards and specifications related to the Event Architecture is provided.

1.2 Document contributor contact points

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

Name	Organization
Angela Amirault	Compusult Limited
Darko Androsevic	Galdos Systems Inc.
Hans Schoebach	Galdos Systems Inc.
Johannes Echterhoff (editor)	International Geospatial Services Institute (iGSI) GmbH
Leif Stainsby	Galdos Systems Inc.
Panagiotis (Peter) A. Vretanos	CubeWerx
Thomas Everding	University of Muenster - Institute for Geoinformatics

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2010-03-04	0.1.0	JE	initial document	created initial draft of the ER
2010-05-27	0.6.0	JE	all	worked on complete document
2010-06-08	0.6.0	JE	all	finalized document

1.4 Future work

This report represents the continuation and refinement of the work that was performed in OWS-6 with respect to developing the OGC Event Architecture. Significant progress has been made in OWS-7. At the same time, many work items were identified to be addressed in the future:

- **Investigate additional Publish/Subscribe Technologies** – WS-Notification was the only technology that could be investigated in detail as to its applicability to realize the general publish / subscribe requirements defined during the testbed. Additional technologies like Atom / AtomPub as well as WS-Eventing need to be investigated in detail as well to enable a decision which technology the OGC should recommend for realizing publish / subscribe functionality per service binding (KVP, POX, WS-*, RESTful). Keep in mind that if no recommendation is made for one technology per binding, interoperability of systems using and offering publish / subscribe functionality is going to be hard to achieve. This work may also lead to a refinement of the requirements model established in OWS-7 and thus also to a refinement of the publish / subscribe behavior model.

- **Detailed Testing of Event Channels** – The OWS-7 Aviation and SFE threads did not make use of dedicated event channels. Subscriptions targeted the whole set of events available at an Event Service instead of connecting to channels that pre-filter the events. The FDF-Geosynchronization thread used some specific event channels and also tested ad hoc channel functionality to a certain extent. Significant performance benefits can be achieved by defining and applying event channels, which is described in this report. Further work should concentrate on defining the concept of event channels in more detail and actually test it. This work should be a cross thread activity to prevent duplicate efforts across future testbed threads. Tools could be developed to simplify the interactions required to create event channels and to manage them.

- **Notification Delivery Mechanisms** – In the testbed, notifications were pushed or pulled from event sources and thus delivered to (subscription) consumers. Sometimes single events were sent when they occurred, other times they were retrieved in regular intervals. Sometimes a notification contained only one event, sometimes multiple events were delivered in one notification. These different ways to deliver events should be investigated in more detail and clearly documented. Batch transmission of multiple events and in pre-defined intervals could significantly reduce the communication overhead of sending single events. Policies could be defined to be used by subscribers to define how exactly events should be delivered to a subscription's consumer.
  This could also incorporate transformation of a given event into another encoding, for example placing the event into an EDXL-DE or CAP message to be delivered via emergency messaging middleware. In that respect, the Web Notification Service could be enhanced to easily send emergency messages to such messaging systems.

- **Improve Event Filtering** – Various aspects of event filtering are described in this report. Issues but also ideas to significantly improve filtering of events have been documented. Especially spatial filtering using dynamic filter properties / event processing functionality could benefit the filtering results. In addition, well-defined structures and mechanisms to enable automatic conversion / transformation of the reference systems used by filter operands and filtered properties would be a valuable feature.
  Further work on these aspects could considerably improve the filtering functionality available at OGC services, especially when event filtering is concerned. Performed as a cross thread activity, this work would benefit multiple domains and avoid duplicate work.

- **Event Security** – Initial work has been performed to identify security threats that apply specifically to an Event Architecture. Measures to mitigate these threats have briefly been described as well. Further work on Event Architecture in general and especially Event Service developments should always take into account relevant security aspects. Work on event security should continuously be improved and actually be tested to achieve the building blocks to deploy an enterprise ready Event-Driven SDI.

- **Specification Guidelines** – Work that was started in OWS-6 to support users (implementers and standards architects) of the Event Architecture was continued in OWS-7. Tutorials how to use Event Architecture components as well as guidelines that define which information is needed in standards that want to include Eventing Architecture functionality (publish / subscribe interfaces, event channels etc) form one important building block of the OGC Event Architecture. Further work on aspects of the Event Architecture should always include the continued development and refinement of tutorials and guidelines for using and deploying the architecture.

- **Event Discovery**

  - **Event information acquisition** – In OWS-7 the TopicSet data type from OASIS WS-Notification was used to provide eventing information for an Event Service. We need to determine how best to acquire a TopicSet document from an Event Service. One option is to make use of WS-Notification accessor methods to get a service's TopicSet . Another possible way to obtain the TopicSet / eventing information would be to put it into a service's Capabilities document (like the SWE 2.0 services intend to do it). Depending on which publish / subscribe interface is going to be used in a given OWS binding, the event information may also be provided in a different way. These approaches have yet to be tested and thus are potential areas for follow-on investigation.

  - **Event channel scheme** – Consider whether discovery is better served by introduction of a classification scheme to identify all event channels. This may be useful if a set of channels is not large and fairly stable. Use of a scheme would make them more accessible to client software and offers

alternative discovery mode: browsing. Also, better documentation of known channels could be achieved. Currently, event channels are matched by "name" (see EventChannel / Name) which may be weak (fragile matching of text strings) compared with the precise, discoverable and extensible use of a scheme.

- **Event type scheme** – As with event channel sets, consider whether event types could usefully be codified as a scheme.

- **Discovery of Channels/Events via Spatial Query** – While not demonstrated during the OWS-7 project, it is possible to associate geometry with any RegistryObject, including EventChannels and EventTypes. This would support searches for channels or events in a registry using a spatial query. To facilitate this, a representative Extent (e.g. gml:Polygon) could be added to each target object (via rim:Slot/wrs:ValueList/wrs:AnyValue property) and then queries could use a spatial operator. The according extent information would need to be available in the encoding of events / event channels.

- **Support multiple aliases on event channel objects** – As event channels may be used by (i.e. referenced from) multiple different application domains, each of which may refer to the particular channel via a distinct label or name, it may be useful to consider adding support for multiple aliases to the discovery model. This would primarily be for event channel objects but may also be valid for event types as well.

- **Complex Spatial Metadata** – Expanding on the *Discovery of Channels/Events via Spatial Query* work item where channels and events may have a single geometry property that identifies their area of applicability, the possibilities of the spatial extent of a channel/event being a multi-geometry should be explored.

1.5 Foreword

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

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

2 References

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

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

NOTE This OWS Common Specification contains a list of normative references that are also applicable to this Engineering Report.

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

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

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

OGC 09-031r1, *OWS-6 SWE Information Model Harmonization ER*

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

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

OGC 10-061, *OWS-7 Dynamic Sensor Notification Engineering Report*

OGC 10-069, *OWS 7 Engineering Report - GeoSynchronization service*

OGC 10-079, *OGC® OWS-7 Aviation Architecture Engineering Report*

OASIS *Web Services Base Notification 1.3*

OASIS *Web Services Brokered Notification 1.3*

OASIS *Web Services Topics 1.3*

3 Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Specification [OGC 06-121r3] and clause 3 of the OWS-6 SWE Event Architecture Engineering Report [OGC 09-032] shall apply. In addition, the following terms and definitions apply.

3.1
publication
Metadata on events generated by a publisher.

3.2
registration
publisher registration
Representation of the relationship between a publisher and consumer; the publisher is entitled to send events to the consumer as long as the registration exists; related to *publication* in the sense that when a registration is created, a publisher may provide metadata on the events it is going to publish.

3.3
subscription
Expression of interest in a specific set of events that are published by a producer.

4 Conventions

4.1 Abbreviated terms

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
AIXM	Aeronautical Information Exchange Model
ASF	Atom Syndication Format
Atom	Atom Syndication Format
AtomPub	Atom Publishing Protocol
BBOX	Bounding Box
CA	Certificate Authority
CAP	Common Alerting Protocol
CSW	Catalogue Service Web
DES	Data Encryption Standard
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
ebRIM	Registry information model specification
ED-SDI	Event-Driven Spatial Data Infrastructure
EML	Event Pattern Markup Language
FAA	Federal Aviation Administration
FDF	Feature Decision Fusion
FES	Filter Encoding Specification
GeoXACML	Geospatial eXtensible Access Control Markup Language
GML	Geography Markup Language
GSS	Geosynchronization Service
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IPSec	Internet Protocol Security
JMS	Java Message Service
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MPLS	Multiprotocol Label Switching
NAT	Network Address Translator

NOTAM	Notice To AirMen
O&M	Observations & Measurements
OASIS	Organization for the Advancement of Structured Information Standards
OWS	OGC Web Service
PKI	Public Key Infrastructure
POX	Plain Old XML
PSHB	PubSubHubBub
pubsub	publish / subscribe
RFC	Request For Comment
SAML	Security Assertion Markup Language
SAS	Sensor Alert Service
SensorML	Sensor Model Language
SES	Sensor Event Service
SFE	Sensor Fusion Enablement
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SPS	Sensor Planning Service
SSL	Secure Sockets Layer
SWE	Sensor Web Enablement
SWG	Standards Working Group
TLS	Transport Layer Security
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WAN	Wide Area Network
WFS	Web Feature Service
WNS	Web Notification Service
WS-E	Web Services Eventing
WS-N	Web Services Notification
WXXM	Weather Information Exchange Model
XACML	eXtensible Access Control Markup Language

XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
XPath	XML Path Language
XPointer	XML Pointer Language
XQuery	XML Query Language

5 Event Architecture – Overview

The Event Architecture addresses the realization of an Event-Driven Spatial Data Infrastructure (ED-SDI). An ED-SDI is a traditional SDI where services and clients also communicate by means of events. This leads to improved (re-) activity of the whole infrastructure which is important if timely communication of information is critical – which is the case for example in emergency scenarios.

This report picks up the work done in OWS-6 and revises it where required - but primarily it extends it. The report brings together results and lessons learned across all OWS-7 threads for event architecture, pubsub and notifications. Previous, ongoing and future work (like on Atom, AtomPub, SAS, SES, WNS, WS-N and CAP) is taken into account and discussed in light of developing a Best Practices for using events and notifications within OGC.

A major part of this report is concerned with aspects related to the realization of a common approach on publish/subscribe (pubsub) for OGC Web Services (OWS). This is a key factor for enabling an ED-SDI. The OWS-6 report (OGC 09-032) defines general interfaces and service roles but does not go into detail on pubsub behavior or parameter models. This report provides missing information and discusses general pubsub functionality. It does so by first developing an abstract model for pubsub in OWS. Then functional requirements are developed based upon the model. Finally, a mapping is performed for a given publish / subscribe technology which shows how well the technology supports the requirements.

This work lays the foundation for the decision which publish / subscribe technology to use in OWS. When only one technology is used per service binding and if different technologies chosen for different bindings support the same publish / subscribe functionality, interoperability can truly be achieved with respect to publishing, subscribing for and consuming events.

6 Publish / Subscribe Functionality

6.1 Introduction

The OGC experience with publish / subscribe technologies showed that various standards and specifications exist (see chapter 12) with which publish / subscribe functionality can be realized. These technologies differ in the way the functionality is provided. Some of them also provide enhanced functionality that is useful in general but usually not in the scope of the other technologies.

Experience in the development of the Sensor Alert Service (SAS) specification, Sensor Event Service (SES) and the Sensor Web Enablement 2.0 standards, especially the Sensor Planning Service (SPS) 2.0 showed that users of OGC standards prefer to reuse existing publish / subscribe technology. Re-design and repeated definition of publish / subscribe interfaces in specific OGC services is undesired and should be avoided. Instead, existing standards should be leveraged if possible.

The variety of existing publish / subscribe standards and specifications as well as the differing amount of functionality they offer make it difficult to decide for one particular technology. As resource oriented service models, also known as RESTful services, have gained ground in the OGC and as these usually have a quite different interface concept than the traditional operation based services[1] the decision for one particular publish / subscribe technology is even more difficult.

These considerations were the drivers to take a step back and consider which publish / subscribe functionality OGC Web Services (OWS) require and which functionality is of interest. In other words we were interested in core publish / subscribe functionality and possible extensions. The functionality was investigated from an abstract point of view, meaning that the same functionality requirements apply regardless which service model / binding is actually used. The mapping of a given publish / subscribe technology to the abstract model, realized through a table that evaluates how well the technology supports the requirements, would then provide criteria to support the decision which pubsub technology should be leveraged in OWS. The goal is to find one technology per OWS binding / architectural style to realize required publish / subscribe functionality.

In OWS-7 publish / subscribe service functionality was realized in multiple threads: FDF Geosynchronization, SFE and Aviation. Testbed participants from each thread contributed to the development of an abstract publish / subscribe service model and the according requirements. Functionality that was commonly or only partly realized in the testbed as well as further functionality was documented using Unified Modeling Language (UML) principles. The functional requirements were also documented using UML.

---

[1] A Web Feature Service is a well-known OGC web service that is currently operation based, meaning that it uses XML encoded request / response messages to invoke service operations. Such service models can easily be integrated into a SOAP / WS-* environment

In the following, the abstract publish / subscribe service model is documented. Both the overall structure as well as behavior are described. Note that the UML model documentation is given in a high level of detail so that it can be used as the foundation for future work, for example an Event Service Standards Working Group (SWG).

Finally, the technology mappings onto these requirements are described.

6.2 Abstract Publish / Subscribe Model

6.2.1 Service Model

The following sections define the interfaces and types that are relevant for an OWS that implements publish / subscribe functionality – the set of functionality defines an OGC Event Service.

Keep in mind that this is an abstract model, not designed with direct implementation in mind. The goal is to have one common service model that can be mapped to given realization technologies (like Atom feeds / WS-Notification and how these are used).

Several packages can be identified – see the following figure.

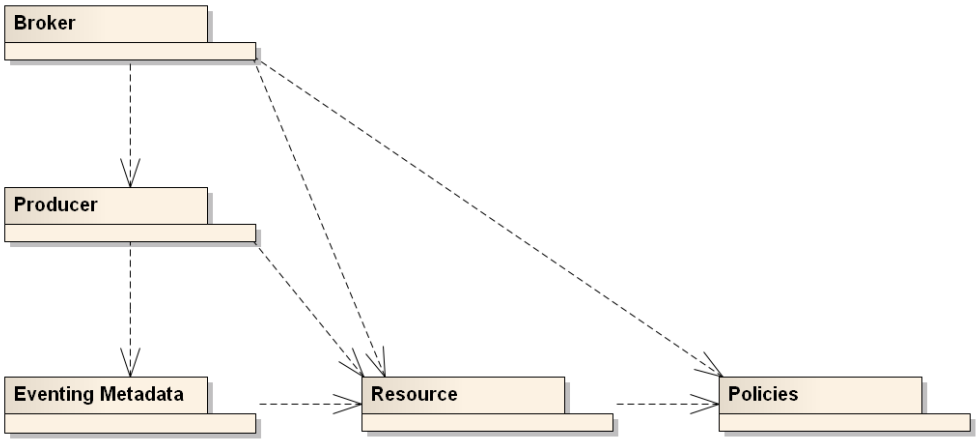

Figure 1: Event Service Package Dependencies

These packages are described in the following.

6.2.1.1 Policies Package

This package defines Event Service Policies - Figure 2 provides a detailed overview. They are used to represent behavior that is offered by / requested from the service.

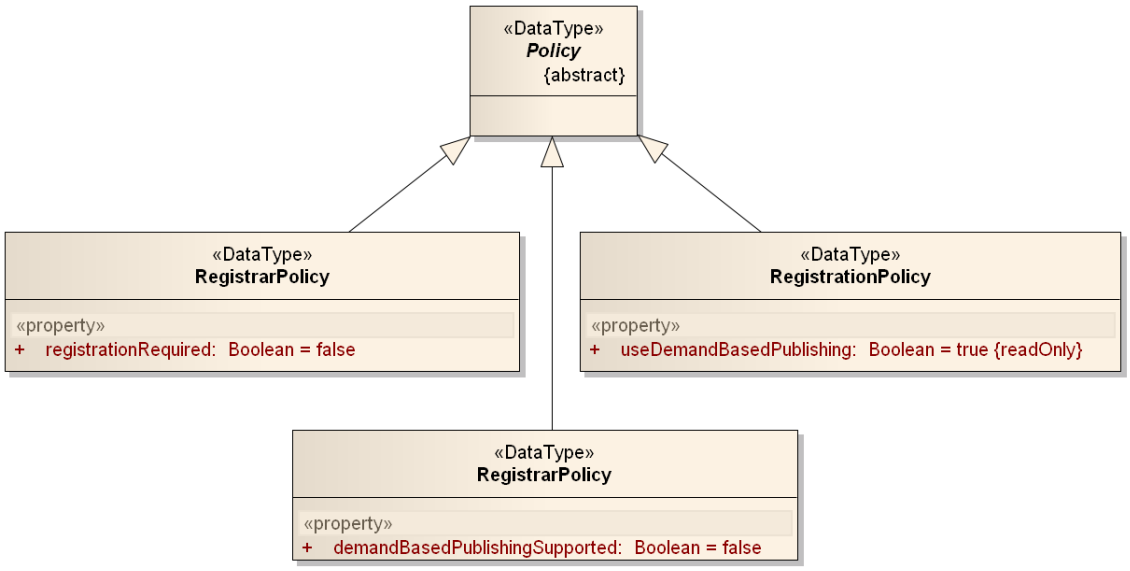

Figure 2: Policy types

The policy types with their properties are described in the following

6.2.1.1.1 Policy Class

Defines executed, executable, requested or requestable behavior.

6.2.1.1.2 RegistrarPolicy Class

Defines registrar behavior.

Table 1 – RegistrarPolicy Properties

Property name	Description	Value
registrationRequired	Determines if the registrar requires that publishers have to be registered before they can send notifications to it.	*Type*: Boolean *Default:* false *Cardinality*: [1]

6.2.1.1.3 RegistrarPolicy Class

Defines broker specific behavior.

Table 2 – RegistrarPolicy Properties

Property name	Description	Value
demandBasedPublishingSupported	Determines if the broker supports the *demand based publication* mechanism.	*Type*: Boolean *Default:* false *Cardinality*: [1]

6.2.1.1.4 RegistrationPolicy Class

Defines behavior associated with a registration.

Property name	Description	Value
useDemandBasedPublishing	When a *Registrar* requests that a *Broker* uses demand based publishing, it requests that the Broker creates a *PausableSubscription* at the *Publisher* referenced in the *Registration* (which then shall be a *Producer*) that is paused whenever the Broker has no subscriptions for the new publication. See the behavior documentation for further details on demand based publishing (section 6.2.2.7).	*Type*: Boolean *Default:* true *Cardinality*: [1]

6.2.1.2 Resource Package

This package contains the abstract *Resource* classes / types. Figure 3 provides a detailed overview.

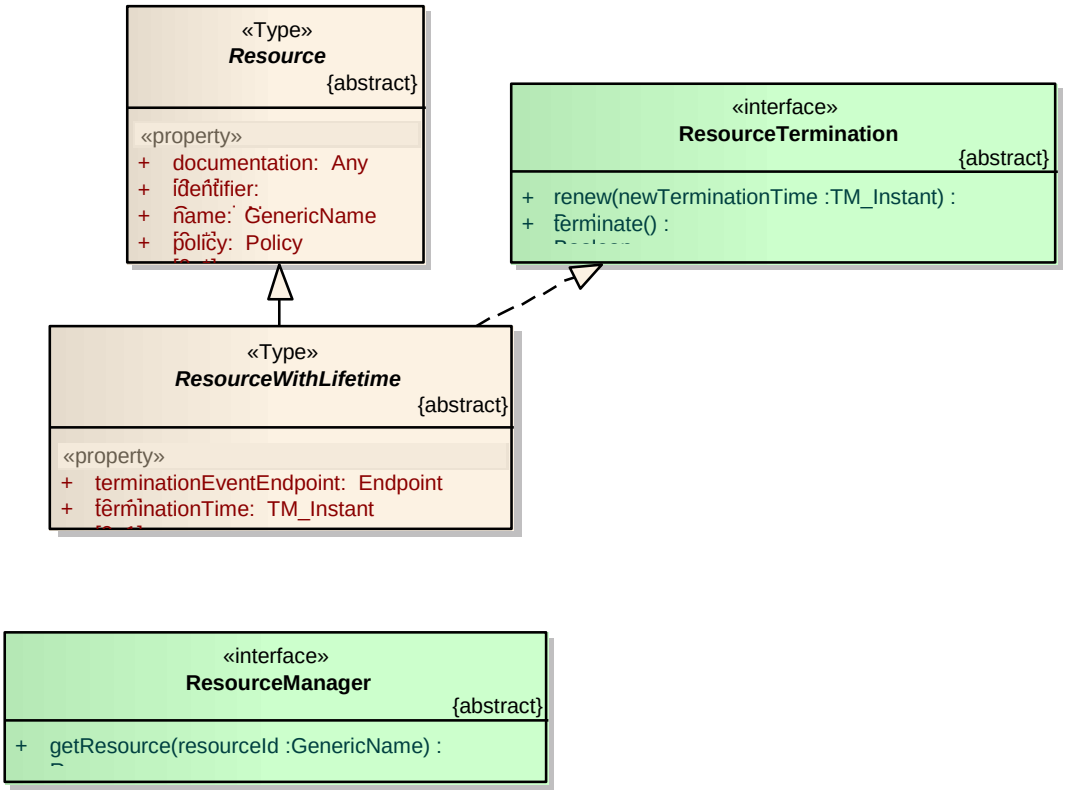

Figure 3: Resource types and interfaces

The resource interfaces and types with their properties are described in the following.

6.2.1.2.1 Resource Class

A resource is an identifiable entity. It may have several properties that can be accessed. In addition, it may execute behavior and may have state.

Table 3 – Resource Properties

Property name	Description	Value
documentation	Descriptive information about the resource (object).	*Type*: Any *Default: <none>* *Cardinality*: [0..1]
identifier	Identifier of the resource object.	*Type*: GenericName *Default: <none>* *Cardinality*: [1]
name	Name(s) assigned to the resource (object). Multiple names may be assigned to the same resource.	*Type*: GenericName *Default: <none>* *Cardinality*: [0..*]
policy	Defines behavior executed by the resource or available / required when interacting with the resource. See *Policies* package for further details.	*Type*: Policy *Default: <none>* *Cardinality*: [0..*]

6.2.1.2.2 ResourceWithLifetime Class

Resources may have a defined lifetime. A resource with lifetime may automatically terminate at a defined point in time or it may explicitly be terminated by another entity. A *ResourceWithLifetime* implements the *ResourceTermination* interface.

A ResourceWithLifetime is a Resource and therefore inherits all properties that a Resource has.

Table 4 – ResourceWithLifetime Properties

Property name	Description	Value
terminationEvent Endpoint	Endpoint of the entity that shall be notified by the producer when the subscription terminated.	*Type*: Endpoint *Default: <none>* *Cardinality*: [0..1]
terminationTime	Absolute point in time when the resource terminates. Termination time is often requested and computed based upon the current server time and a duration - this can be mapped to one TM_Instant.	*Type*: TM_Instant *Default: <none>* *Cardinality*: [0..1]

6.2.1.2.3 ResourceManager Interface

A system entity that provides access to resources is a *ResourceManager*. This interface defines the resource management operation(s).

Table 5 – ResourceManager Operation

Method	Notes
getResource	Retrieves a resource via its identifier.

6.2.1.2.4 ResourceTermination Interface

The *ResourceTermination* interface defines operations to reset the termination time of a resource with defined lifetime and to explicitly terminate such a resource.

Table 6 – ResourceTermination Operations

Method	Notes
renew	Resets the termination time of the resource to a given time instant. The resource may reject the request and send an exception, for example if the proposed termination time does not conform to the resource's termination time policies. Setting the termination time to a value in the past will - if the request is accepted - lead to the resource's termination according to its implementation (when a resource has to check its termination is not defined; however, it is recommended that the implementation terminates a resource shortly after the termination time passed).
terminate	Immediately terminates the resource.

6.2.1.3 EventMetadata Package

This package defines types / classes that capture eventing metadata. The data is useful for discovering information relevant for publish / subscribe systems. Primarily the definition of event types as well as event channels together with their relationships can be modeled and discovered (see chapter 7).

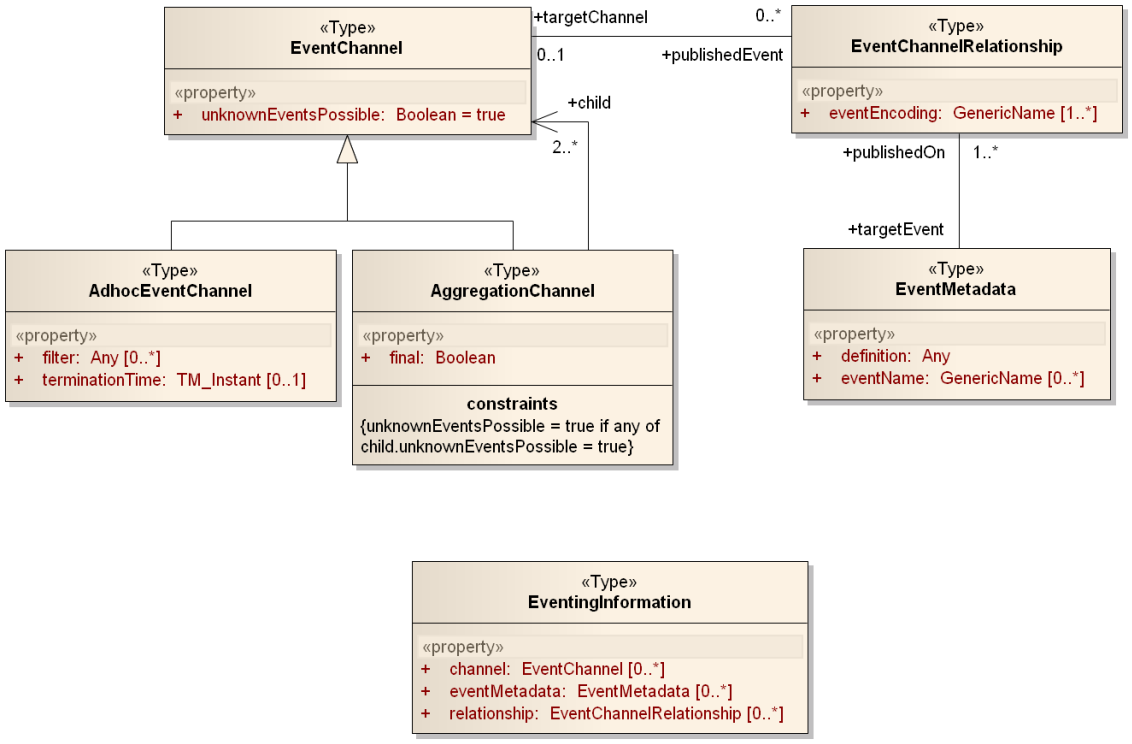

Figure 4: EventMetadata types

The event metadata types with their properties are described in the following.

Note: The XML Schema implementation of this package is provided in chapter 16, *Annex D – Event Metadata* .

6.2.1.3.1 AdHocEventChannel Class

Represents an *EventChannel* that was created ad hoc as additional delivery target for a subscription.

Other subscribers can explore existing ad hoc channels and which events they target (via the filter statements and documentation) and subscribe for the same events. Handling subscriptions that target the same events is thus made easier for both subscribers and the producer.

Subscribers to this kind of channel should be aware that the channel may terminate any time (because the underlying subscription may be terminated at any time). A given termination time is therefore only indicative.

An AdHocEventChannel is an EventChannel and therefore inherits all properties that an EventChannel has.

Table 7 – AdHocEventChannel Properties

Property name	Description	Value
filter	Filter statement used by the subscription that created the ad-hoc channel.	*Type*: Any *Default: <none>* *Cardinality*: [0..*]
terminationTime	Time when the channel is set to automatically expire. This shall be the same as the termination time of the subscription that created the ad hoc channel. Note: the termination time - if given - is only indicative, as the underlying subscription may be terminated at any time.	*Type*: TM_Instant *Default: <none>* *Cardinality*: [0..1]

6.2.1.3.2 AggregationChannel Class

Represents a channel in which all events from the child channels are published as well. This makes it easier for clients to subscribe for a number of channels.

An AggregationChannel is an EventChannel and therefore inherits all properties that an EventChannel has.

Table 8 – AggregationChannel Properties

Property name	Description	Value
final	true if the aggregation channel's set of child channels does not change (children are removed or added) - false otherwise	*Type*: Boolean *Default: <none>* *Cardinality*: [1]
child	An AggregationChannel has two or more child channels.	*Type*: EventChannel *Default: <none>* *Cardinality*: [2..*]

6.2.1.3.3 EventChannel Class

An EventChannel is a Resource. Multiple events may be published on the channel. In some cases it may not be possible to identify these events.

Table 9 – EventChannel Properties

Property name	Description	Value
unknownEventsPossible	True if more than the listed events may be published on this channel, otherwise false.	*Type*: Boolean *Default:* true *Cardinality*: [1]
publishedEvent	Various events may be published on one channel.	*Type*: EventChannelRelationship *Default: <none>* *Cardinality*: [0..*]

6.2.1.3.4 EventChannelRelationship Class

Represents the association class of the association that exists between an EventChannel and EventMetadata.

Note: the association class has been modeled explicitly here as it might be encoded as an XML implementation using GML Application Schema encoding rules in the future. Then the metadata of an Event Service could include this metadata.

Table 10 – EventChannelRelationship Properties

Property name	Description	Value
eventEncoding	Encoding of an event for a given channel. Note: the property may be null in cases in which the exact encoding of an event cannot be determined. In the XSD implementation this should be indicated using the nilReason attribute.	*Type*: GenericName *Default: <none>* *Cardinality*: [1..*]
targetChannel	Various events may be published on one channel.	*Type*: EventChannel *Default: <none>* *Cardinality*: [0..1]
targetEvent	An event is published in certain encodings on certain channels. An event is published at least on the pre-defined "all-event" channel.	*Type*: EventMetadata *Default: <none>* *Cardinality*: [1]

6.2.1.3.5 EventMetadata Class

Represents semantics of an event.

Table 11 – EventMetadata Properties

Property name	Description	Value
definition	Defines the event in as much detail as possible. Any type of definition is suitable, from a pure textual to a pure ontology based definition.	*Type*: Any *Default: <none>* *Cardinality*: [1]
eventName	Name of the event - rather, the happening it represents - assigned by a given domain. Note: In the future we may consider using GFI_Feature here, given byReference - and the xlink:href provides the identifier for the actual event feature type - much like it is currently done with the observedProperty in an O&M observation.	*Type*: GenericName *Default: <none>* *Cardinality*: [0..*]
publishedOn	An event is published in certain encodings on certain channels. An event is published at least on the pre-defined "all-event" channel.	*Type*: EventChannelRelationship *Default: <none>* *Cardinality*: [1]

6.2.1.3.6 EventingInformation Class

This type serves as the container to describe EventChannel / EventMetadata and their relationships.

Table 12 – EventingInformation Properties

Property name	Description	Value
channel	A channel supported by the event service.	*Type*: EventChannel *Default: <none>* *Cardinality*: [0..*]
eventMetadata	Metadata about a type of event that is published by the service.	*Type*: EventMetadata *Default: <none>* *Cardinality*: [0..*]
relationship	The relationship that explains the encoding being used to publish a given type of event on a given channel.	*Type*: EventChannelRelationship *Default: <none>* *Cardinality*: [0..*]

6.2.1.4 Producer Package

This package contains the classes / types relevant for a *Producer*. The following diagrams provide an overview.

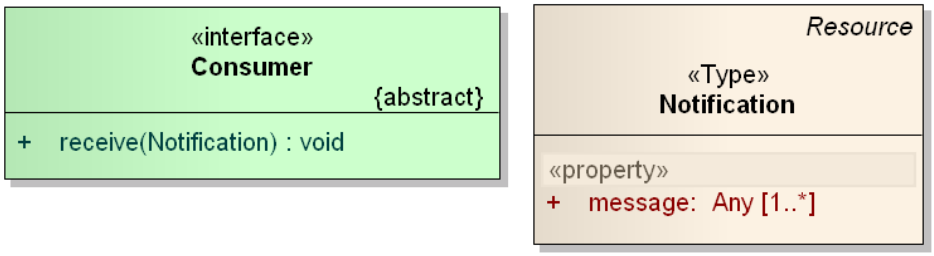

Figure 5: Notification type and Consumer interface

A *Consumer* can receive a *Notification* which contains one or more messages, for example providing information about an event.

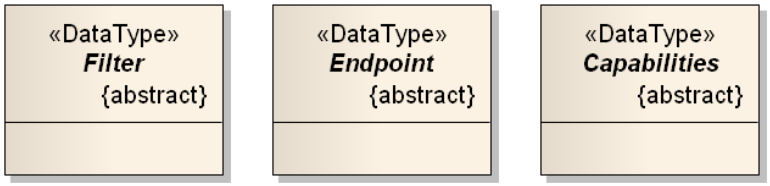

Figure 6: Abstract types used by Producer types

Figure 6 lists the generic types used in the Event Service model. These types represent placeholders for realization through specific types / technologies. For example, a Filter may be realized through an OGC Filter Encoding Specification (FES) filter statement.

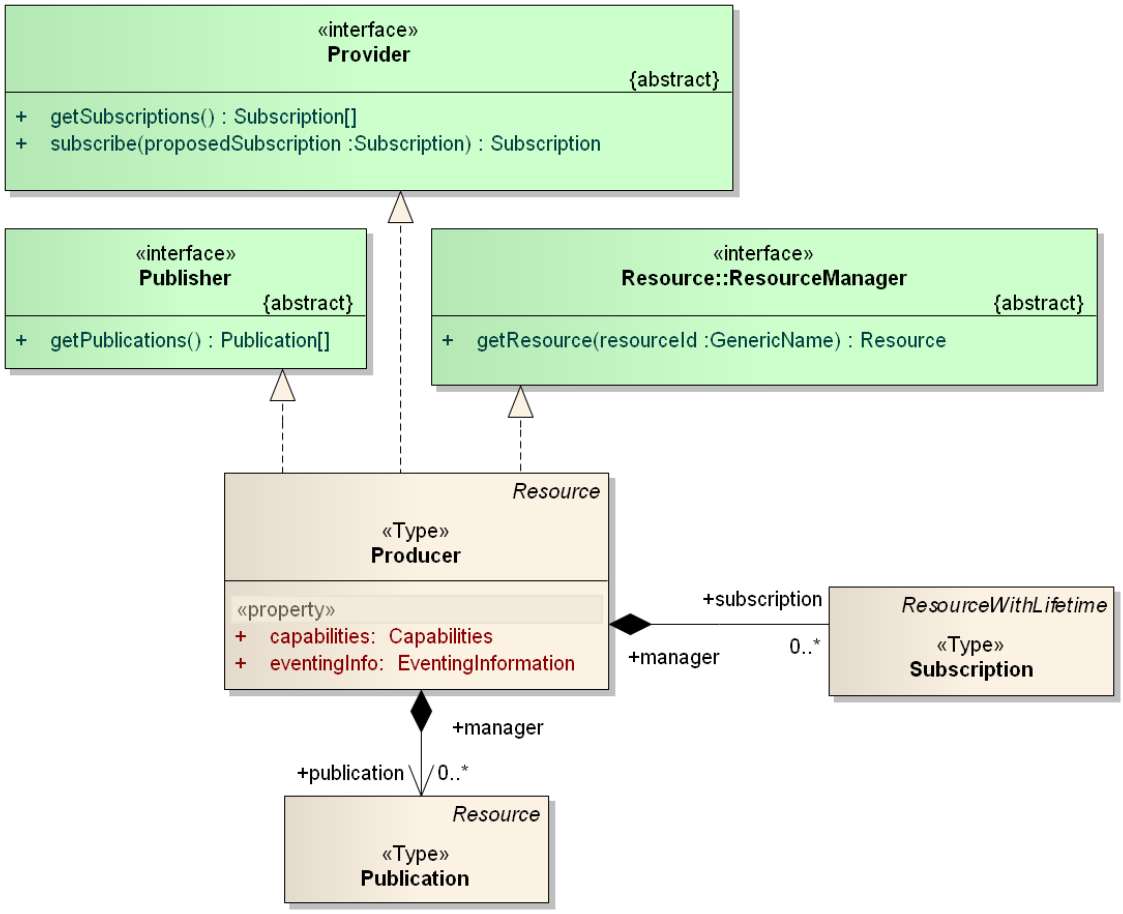

Figure 7: Producer types and interfaces

A gigh-level overview of a Producer is given in Figure 7 – the *Subscription* and *Publication* are depicted in more detail in the following diagrams.

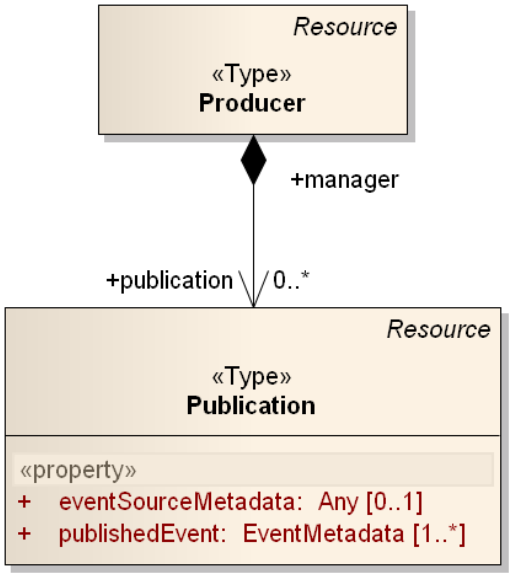

Figure 8: Publication type

Figure 8 provides a detailed view upon the Publication type and associated types while a detailed view upon the Subscription type and associated types is shown in Figure 9.

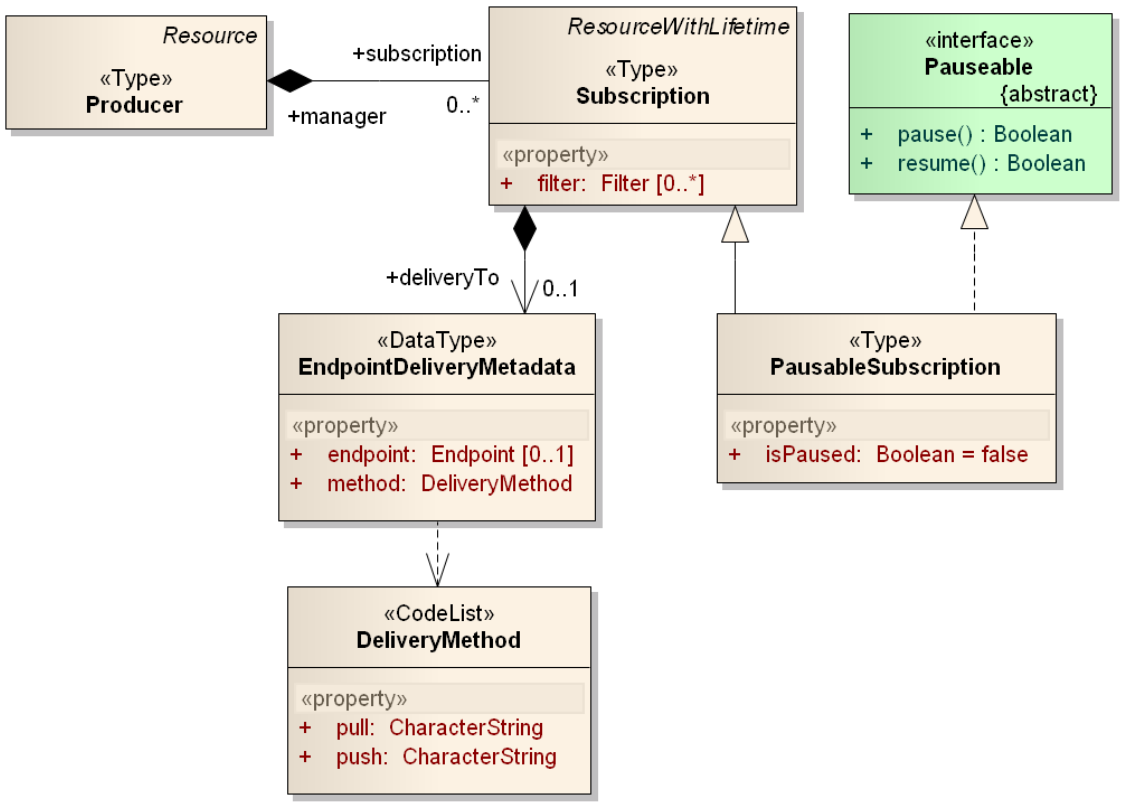

Figure 9: Subscription types

The producer interfaces and types with their properties are described in the following.

6.2.1.4.1 Capabilities Class

Abstract class that represents all types of information about what a producer is actually capable to do.

This may include filter capabilities, support for demand-based publishing, default consumer endpoint for publishers to send notifications to a broker etc.

6.2.1.4.2 DeliveryMethod CodeList

The delivery method identifies how events shall be sent to a delivery endpoint.

Table 13 – DeliveryMethod Properties

Property name	Description	Value
pull	Events shall be provided via an endpoint where clients can pull events from.	*Type*: CharacterString
push	Events shall be pushed to the given endpoint.	*Type*: CharacterString

6.2.1.4.3 Endpoint Class

Metadata to communicate with an entity.

Note: in a (web) service environment this is usually the service address (expressed as a URL or WS-Addressing endpoint). In a programming environment (like a Java program) it may also be an object pointer / reference.

6.2.1.4.4 EndpointDeliveryMetadata Class

Metadata on the consumer endpoint where events matching the subscription shall be sent to.

Table 14 – EndpointDeliveryMetadata Properties

Property name	Description	Value
endpoint	Endpoint where events shall be sent to. Shall be omitted when subscription with pull delivery is proposed so that the service can fill in the endpoint if it accepts the subscription.	*Type*: Endpoint *Default: <none>* *Cardinality*: [0..1]
method	Code for the method used to send notifications.	*Type*: DeliveryMethod *Default: <none>* *Cardinality*: [1]

6.2.1.4.5 Filter Class

Abstract class that represents all types of filter statements.

Note: the filter can be realized for example using XPath expressions, WS-Notification topic expressions, FES filter statements etc.

Note: a filter may be implicit - e.g. if a consumer is subscribed to an Atom feed then the filter to match only events posted on that feed is automatically created.

6.2.1.4.6 Notification Class

Container for messages. Multiple messages that shall be sent at the same time (defined by the producer and/or via policy) may be contained in one notification.

A Notification is a Resource and therefore inherits all properties that a Resource has.

Table 15 – Notification Properties

Property name	Description	Value
message	Piece of information that shall be delivered. Note: the cardinality of this attribute is one to multiple to support batch transmission of messages. This is one way to decrease the amount of communication channels (like HTTP connections) that need to be established between a publisher and consumer. We recognize that some communication protocols can establish persistent connections so that this kind of mechanism would not be required there. Policies could be defined that control the maximum time interval or amount of messages that would be used to gather up messages before sending - this would be a kind of batch transport. Different ways to perform, control and mange notification delivery can be investigated in the future.	*Type*: Any *Default: <none>* *Cardinality*: [1..*]

6.2.1.4.7 PausableSubscription Class

A *PausableSubscription* is a *Subscription* that can be paused.

Pausing a subscription means that events that match the subscription criteria shall not be delivered to the assigned consumer endpoint (while the subscription is paused). This does not affect the delivery of matching events to the (optional) channel delivery endpoint. In other words: if an AdHocEventChannel (see section 6.2.1.3.1) is associated with a subscription then all events matching the subscriptions filter criteria shall be published on that channel, regardless of whether the subscription is paused or not.

All events that are received or generated by the producer while a subscription is active (e.g. if it has been resumed) and that match the filter criteria shall be published to the existing delivery endpoint(s).

A PausableSubscription is a Subscription and therefore inherits all properties that a Subscription has.

Table 16 – PausableSubscription Properties

Property name	Description	Value
isPaused	Indicates whether the subscription is paused or not.	*Type*: Boolean *Default:* false *Cardinality*: [1]

6.2.1.4.8 Producer Class

A Producer is an entity that publishes events and allows clients to subscribe for these events.

A Producer is a Resource that manages Publications, Subscriptions and EventChannels. It implements the ResourceManager interface to provide access to these resources. It realizes the Provider interface which allows the creation of new subscriptions. Finally, it also implements the Publisher interface.

A Producer is a Resource and therefore inherits all properties that a Resource has.

Table 17 – Producer Properties

Attribute	Description	Value
capabilities	Metadata about the producer. The type of this attribute is abstract. The OGC Capabilities document is one possible value for this property.	*Type*: Capabilities *Default: <none>* *Cardinality*: [1]
eventingInfo	Information about the event channels exposed and events published by the service.	*Type*: EventingInformation *Default: <none>* *Cardinality*: [1]
publication	A Producer manages Publications.	*Type*: Publication *Default: <none>* *Cardinality*: [0..*]
subscription	A Producer manages Subscriptions. Subscriptions that are terminated are automatically removed from the Producer.	*Type*: Subscription *Default: <none>* *Cardinality*: [0..*]

6.2.1.4.9 Publication Class

Metadata on events generated by a publisher. A *Publication* is a *Resource*.

A Publication is a Resource and therefore inherits all properties that a Resource has.

Table 18 – Publication Properties

Property name	Description	Value
eventSourceMetadata	Metadata about the entity that generates the published events (e.g. SensorML description for a sensor).	*Type*: Any *Default: <none>* *Cardinality*: [0..1]
publishedEvent	Definition of an event that is published as part of this Publication.	*Type*: EventMetadata *Default: <none>* *Cardinality*: [1..*]

6.2.1.4.10 Subscription Class

Represents the expression of interest (via filters) in a specific set of events that are published by a Producer. If no filter is provided, the subscriber expresses interest in all events published by the Producer.

A subscription may define the endpoint of a consumer to which matching events shall be delivered and/or may define a channel to which all events matching the subscriptions filter criteria shall be delivered. This is useful for other subscribers that are interested in the same set of events. At least one delivery target shall be provided.

A Subscription is a ResourceWithLifetime and therefore inherits all properties that a ResourceWithLifetime has.

Table 19 – Subscription Properties

Property name	Description	Value
filter	Statement to narrow down the amount of matching events. Expresses the interest of the subscriber in certain events. Multiple filter statements are connected via an implicit AND.	*Type*: Filter *Default: <none>* *Cardinality*: [0..*]
deliveryTo	The delivery method and endpoint of the consumer.	*Type*: EndpointDeliveryMetadata *Default: <none>* *Cardinality*: [0..1]

6.2.1.4.11 Consumer Interface

Defines the operation to be used for sending a notification to the implementing entity.

Table 20 – Consumer Operation

Method	Notes
receive	Delivers a message. No response is expected from the consumer.

6.2.1.4.12 Pauseable Interface

Defines operations to pause and resume a system entity. What this means is up to the implementing entity.

Table 21 – Pauseable Operations

Method	Notes
pause	Pauses the entity.
resume	Resumes the entity.

6.2.1.4.13 Provider Interface

Defines operations to subscribe for notifications and to get existing subscriptions.

Table 22 – Provider Operations

Method	Notes
getSubscriptions	Retrieves all currently existing subscriptions.
subscribe	Creates a subscription. Upon success, the (pointer to the) new subscription is returned - otherwise an exception.

6.2.1.4.14 Publisher Interface

Defines an operation to get metadata available by a publisher on the events it publishes.

Table 23 – Publisher Operation

Method	Notes
getPublications	Retrieves metadata available on published events.

6.2.1.5 Broker Package

This package contains the classes / types relevant for a Broker. The following diagrams provide an overview.

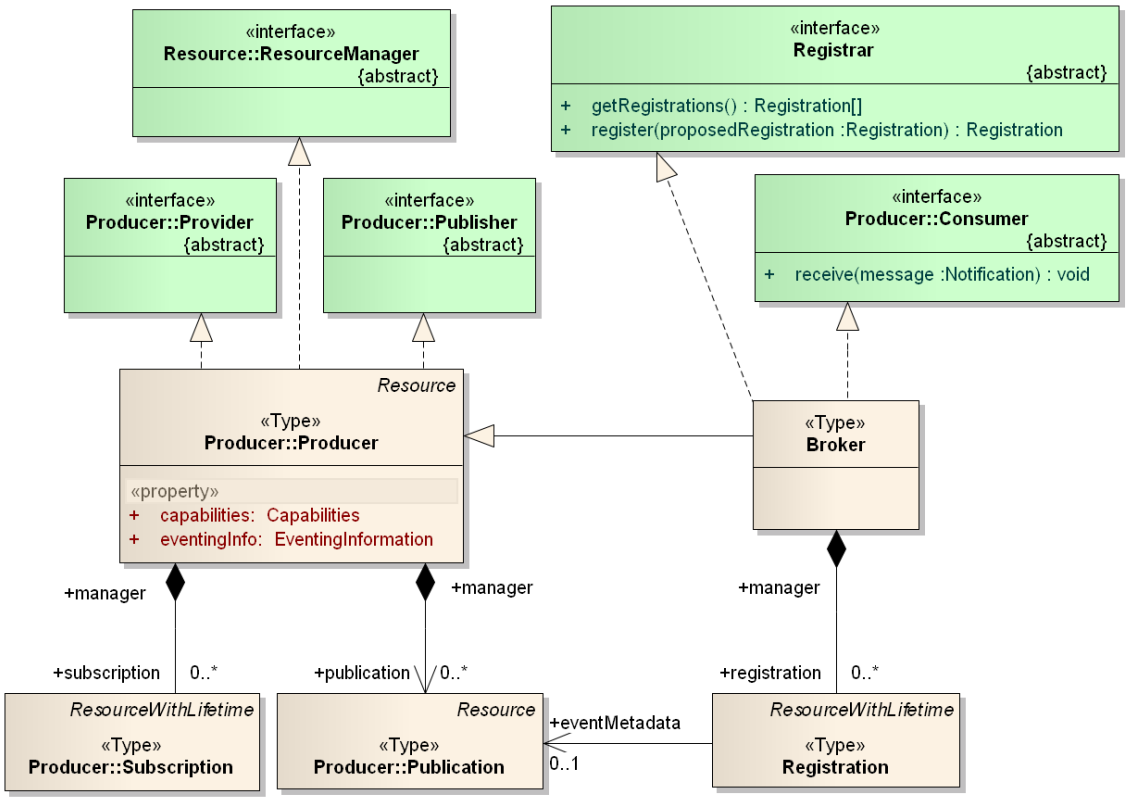

Figure 10: Broker types, interfaces and dependencies to Producer

Figure 10 shows a high-level view upon the Broker and its dependencies.

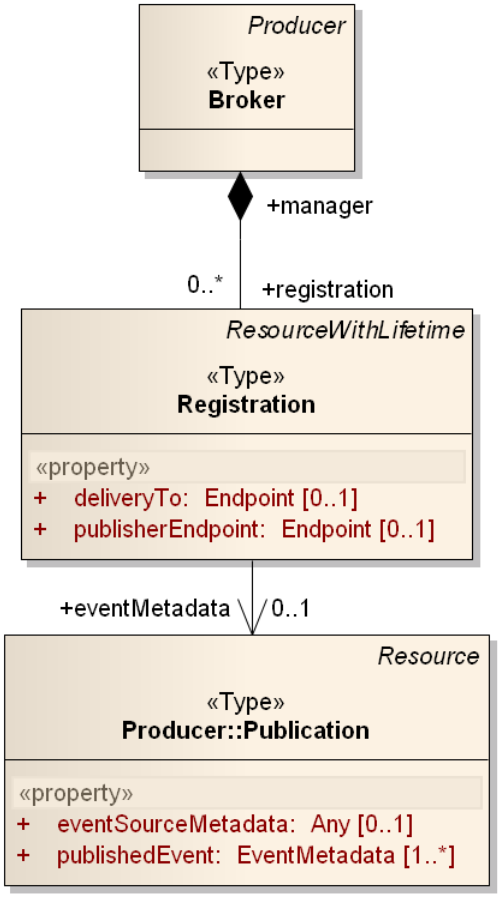

Figure 11: Registration type

Figure 11 provides a detailed view upon the Registration type and associated types.

The broker interfaces and types with their properties are described in the following.

6.2.1.5.1 Broker Class

A broker is a *Producer* and thus offers a Producer's functionality. In addition, a broker implements the *Consumer* interface. It is therefore capable of receiving notifications from publishers that were registered at the broker via the according operation in the *Registrar* interface (which is also implemented by the broker). In addition to subscriptions and publications, the broker manages publisher *Registrations.*

Note: in general, a broker should provide access (via subscriptions) to the notifications received at the brokers Consumer-side. However, a broker is not obliged to grant access to every received notification. Some notifications can be invisible to a subscription because of security constraints or they are just for service internal use.

Table 24 – Broker Properties

Property name	Description	Value
registration	A broker manages registrations.	*Type*: Registration *Default: <none>* *Cardinality*: [0..*]

6.2.1.5.2 Registration Class

Metadata on a registration which is a resource with lifetime and thus may be terminated at any time.

Table 25 – Registration Properties

Property name	Description	Value
deliveryTo	The consumer endpoint of the broker where new notifications for this registration shall be sent to. The delivery endpoint shall be omitted when a registration is proposed to an Event Service. If the service accepts the registration, it shall fill in the endpoint.	*Type*: Endpoint *Default: <none>* *Cardinality*: [0..1]
publisherEndpoint	Endpoint of the entity that publishes events - not necessarily identical to the event source (entity that generated the event). This property is required in a proposed Registration which requests demand based publication behavior. The endpoint then has to reference a *Producer*.	*Type*: Endpoint *Default: <none>* *Cardinality*: [0..1]
eventMetadata	Metadata on the events that are going to be published as part of this registration. May be omitted in case that this information is not available. Note: whether the association is directed or not depends on whether a client should be able to see in a publication who is the publisher responsible for it.	*Type*: Publication *Default: <none>* *Cardinality*: [0..1]

6.2.1.5.3 Registrar Interface

Defines operations to register for sending notifications to the implementing entity and to get existing registrations.

Table 26 – Registrar Operations

Method	Notes
getRegistrations	Retrieves all currently existing registrations.
register	Creates a registration. Upon success, the new registration is returned - otherwise an exception.

6.2.2 Behavior Model

The following subsections document a selection of the behavior that is expected from an OGC publish / subscribe service. The model mostly covers important aspects. This work can and should be continued in the future to define the desired behavior in more detail.

6.2.2.1 Resources Lifecycle Management

As described in section 6.2.1.2, a resource may have a lifetime. If it has, then it can be explicitly terminated. It may also be terminated automatically once the scheduled termination time has expired. The following diagram shows the interactions and events involved in terminating a resource. This lifecycle management applies to all resources with lifetime, for example subscriptions and registrations.

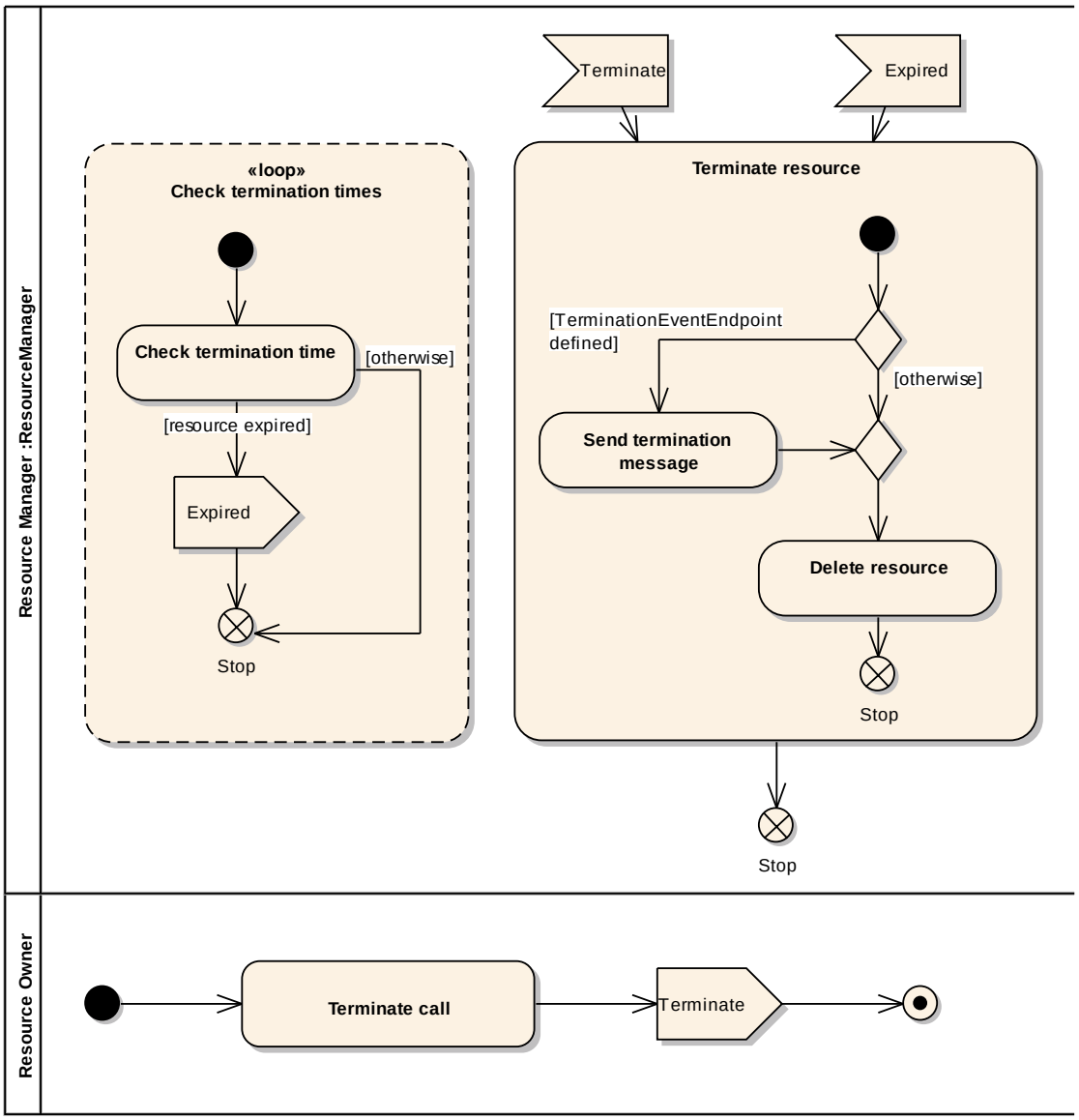

Figure 12: Scheduled and Immediate Resource Termination Acitivity

Figure 12 shows that the activities of the resource manager and owner are performed in parallel.

When a resource owner invokes the terminate operation on a resource, a *Terminate* event is emitted.

The resource manager continuously checks the termination times of all resources with lifetime it governs. If a resource has expired, an *Expired* event is emitted.

When receiving a *Terminate* or *Expired* event, the resource manager terminates the resource that is pointed to by the event. Before deleting a resource, the resource manager has to check if a termination message shall be published. In case that a TerminationEventEndpoint was set for the resource, a termination event is sent to that endpoint.

Figure 13 shows the state machine of a *Resource*. Note that the renew operation does not lead to a state transition but that the expiration of the termination time or an explicit terminate request lead to a transition into the final state.

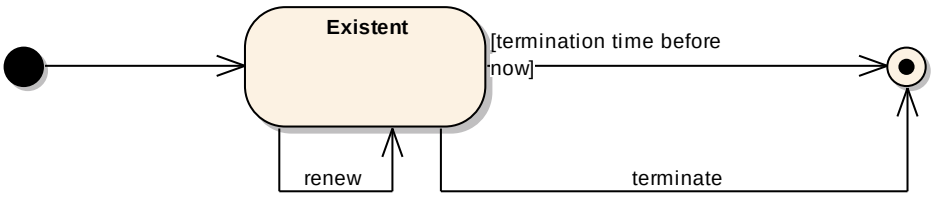

Figure 13: Resource State Machine

6.2.2.2 Managing Subscriptions

Handling subscriptions is one of the most important functionality in a publish / subscribe system. A subscriber may create a subscription at a Producer (see section 6.2.1.4). Figure 14 shows the activities performed and objects created when initializing a new subscription.

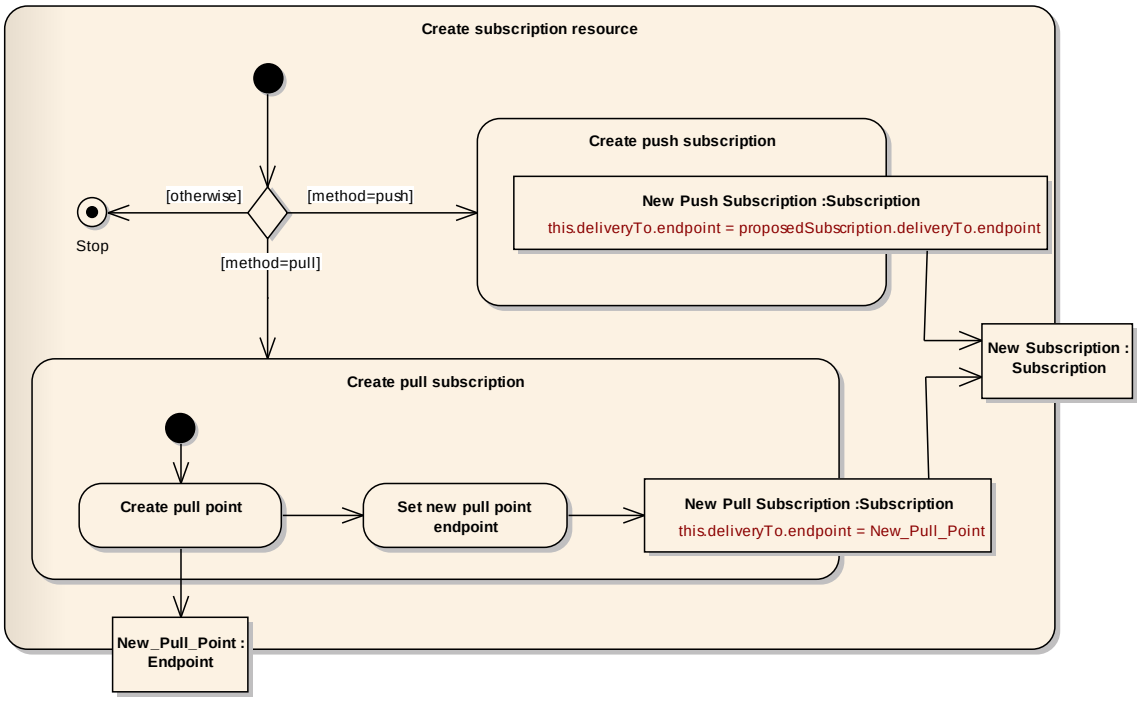

Figure 14: Creating a Subscription – Activity Diagram

Focus is on creating a subscription where the consumer notification is push or pull based.

First of all the provider needs to check the delivery method set for the consumer delivery. Methods currently defined are *push* and *pull*.

If push based delivery is requested, the provider creates a new subscription (object) in which the deliveryTo endpoint is set to the value provided in the proposed subscription.

If pull based delivery is requested, the provider first creates a new pull endpoint and then sets this endpoint as the value of the deliveryTo property of the new subscription (object).

The following diagram shows the interactions involved in creating and managing a subscription.

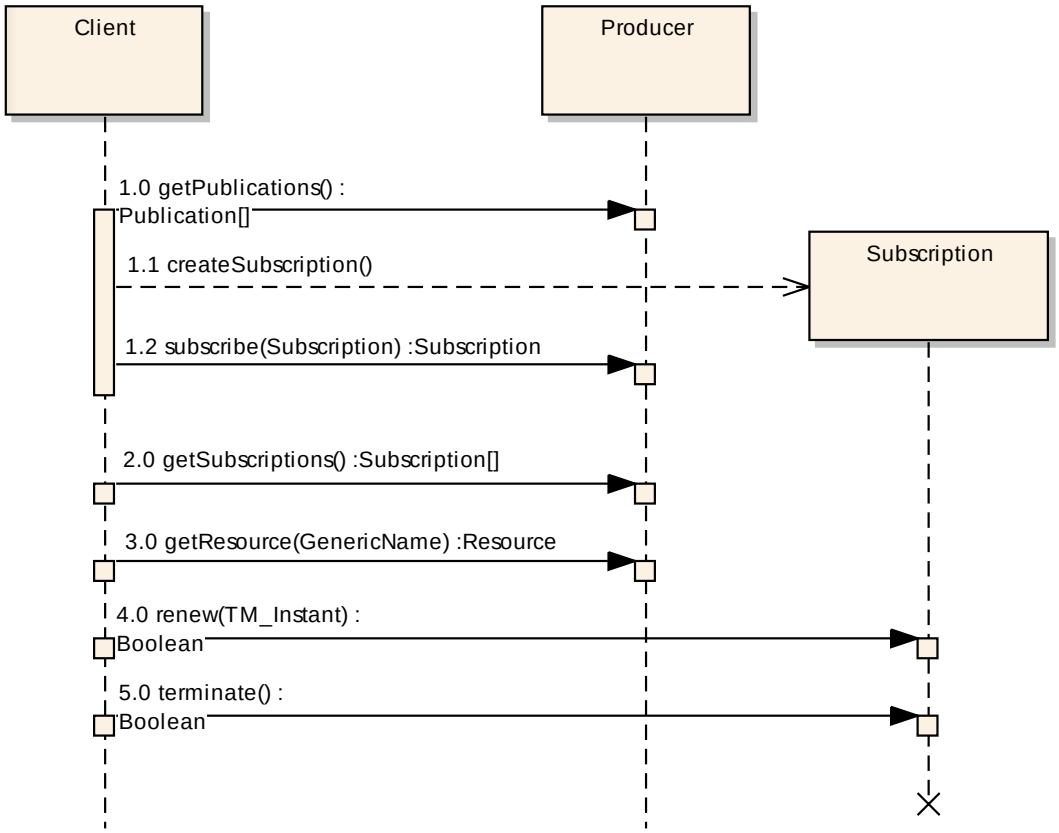

Figure 15: Subscriber Interactions

The following table explains the interactions shown in Figure 15.

Table 27: Explanation of Subscriber Interactions

Message(s)	Notes
Msg: 1.0 *From:* *Client* *To:* *Producer*	The client retrieves the list of all currently existing Publications from the Producer.
Msg: 1.1 *From:* *Client* *To:* *Subscription*	Based upon the information about available Publications, the client creates a new Subscription that is going to be proposed to the Producer. Note: for simplicity only one Subscription is modelled here. In a web service environment, it is more likely that two distinct subscription resources will be created: one governed by the client which is proposed to the producer and one governed by the producer that was created based upon the proposed subscription.
Msg: 1.2 *From:* *Client* *To:* *Producer*	The client subscribes at the Producer by sending a subscription proposal. The Producer sends (a pointer to) the accepted subscription in the response.
Msg: 2.0 *From:* *Client* *To:* *Producer*	The client may retrieve the list of all currently existing Subscriptions from the Producer.
Msg: 3.0 *From:* *Client* *To:* *Producer*	The client may retrieve a Resource hosted by the Producer by its name.
Msg: 4.0 *From:* *Client* *To:* *Subscription*	The client renews the subscription by setting a new termination time.
Msg: 5.0 *From:* *Client* *To:* *Subscription*	The client terminates the subscription.

Figure 16 shows that a Broker only starts matching new events against a subscription when the response to the subscription has been sent to the Subscriber.

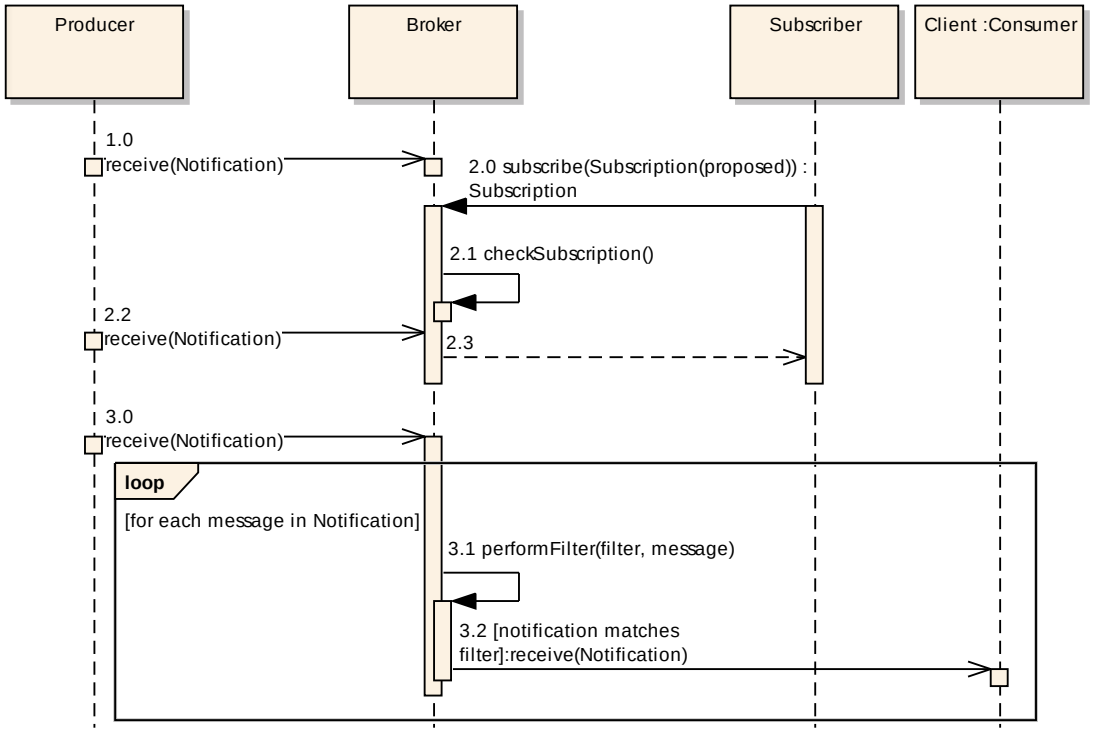

Figure 16: Subscription Timing

The interactions shown in Figure 16 are explained in Table 28.

Table 28: Discussion of Subscription Timing Interactions

Message(s)	**Notes**
Msg: 1.0 *From:* *Producer* *To:* *Broker*	An Event Source sends a notification to the Producer. This will be matched against the currently existing subscriptions at the Broker (not shown in the diagram).
Msg: 2.0 *From:* *Subscriber* *To:* *Broker*	A subscription is proposed to the Producer.
Msg: 2.1 *From:* *Broker* *To:* *Broker*	The Producer checks the subscription to determine if it can be accepted. This can include validity checks or check of given and supported filter statements, for example.
Msg: 2.2 *From:* *Producer* *To:* *Broker*	The Event Source sends another notification to the Producer. The notification is not matched against the subscription yet.
Msg: 2.3 *From:* *Broker*	The subscriber is informed about the outcome of the subscription. In the following, it is assumed that the subscription was accepted.

To: Subscriber	
Msg: 3.0 *From: Producer* *To: Broker*	The Event Source sends a notification to the Producer.
Msg: 3.1 *From: Broker* *To: Broker*	The message is matched against the subscriptions filter criteria. Note: this is done for each message in the notification.
Msg: 3.2 *From: Broker* *To: Client: Consumer*	If the message matches the filter criteria then it is sent as a notification to the subscription's consumer(s).

6.2.2.3 Handling a Pausable Subscription

A publish / subscribe service may support pausable subscriptions. In doing so, it allows clients to pause a subscription while it is not needed. This can save valuable resources both for a producer and a consumer. The following figure shows the state machine of a PausableSubscription.

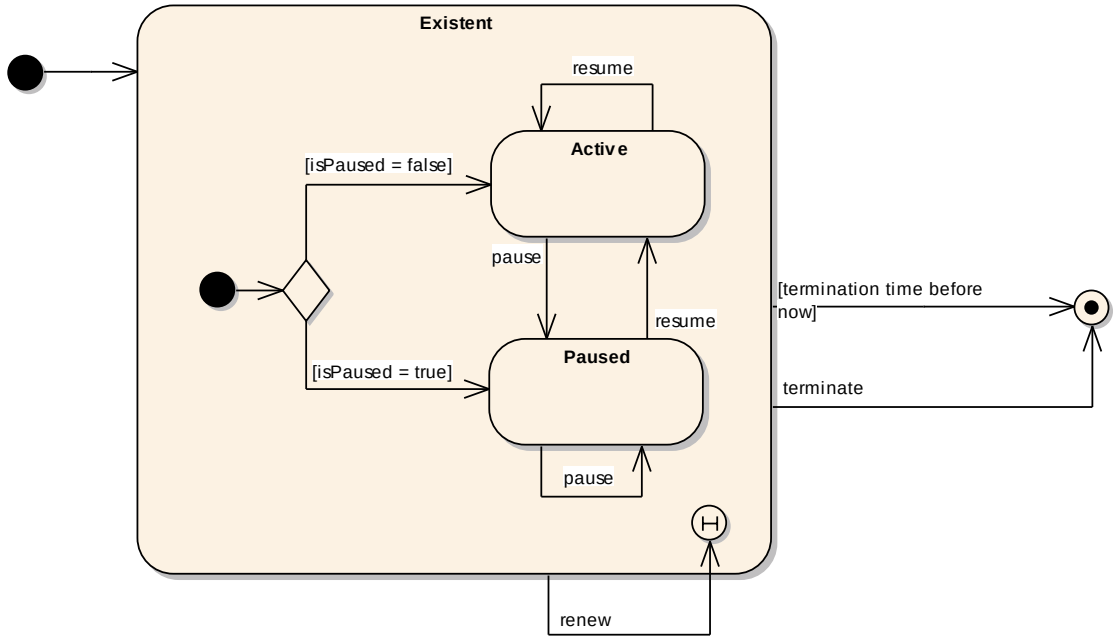

Figure 17: Pausable Subscription State Machine

6.2.2.4 Sending a Notification

One of the most important activities in a publish / subscribe system is to actually send notifications. Figure 18 shows the activity of publishing an event. A Notify event / message is emitted (and sent to some not further specified system entity).

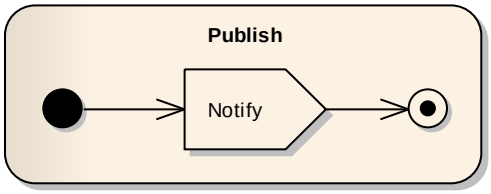

Figure 18: Publishing Activity

The activity relies upon the interface defined in section 6.2.1.4.11.

6.2.2.5 Handling a Notification at a Broker

A broker is capable of receiving incoming notifications and matching them against existing subscriptions. Each notification matching a subscription's filter criteria needs to be published to endpoint defined for that subscription.

The following diagram shows the activities and events involved in handling an incoming notification at a broker. The activities of the publisher and broker run in parallel.

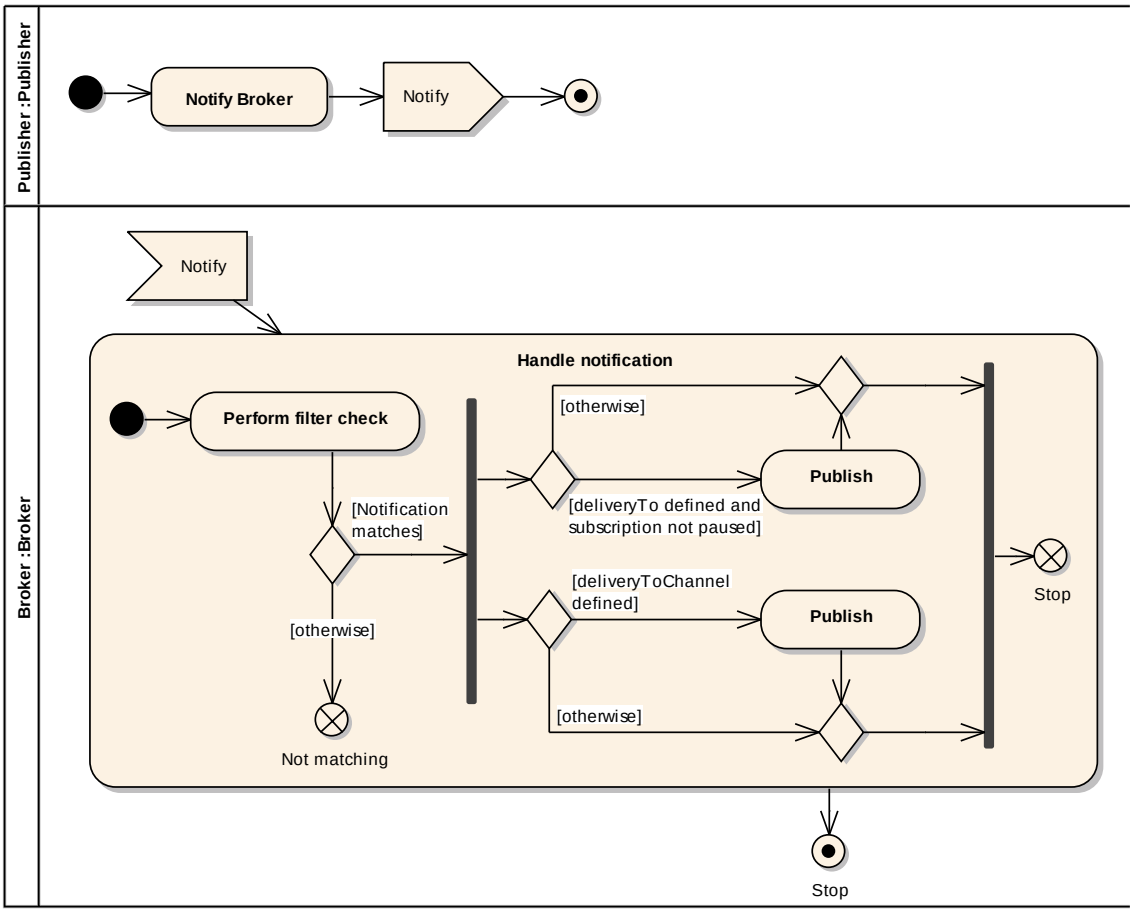

Figure 19: Notification Handling Activity

A publisher notifies a broker by emitting a Notify event / message which is sent to the broker. As no response (which would be a related event) is expected by the publisher, its activity terminates after the notification has been sent.

When receiving a Notify event / message, the broker checks the filter of each subscription it manages against the event. If the event matches, the broker publishes it.

In case that a deliveryTo endpoint is defined and the subscription is not paused, the event is published to the deliveryTo endpoint. If the deliveryToChannel is set in the subscription, the event is published to that endpoint. Note that at least one of these delivery endpoints shall be defined for each subscription. When the matching and publication was performed the handling of the notification for a specific subscription stops.

6.2.2.6 Registering a New Publisher

A new event source can be registered at an Event Service / Broker as new publisher. The interactions involved in creating and managing a registration and publication are shown in the following sequence diagram.

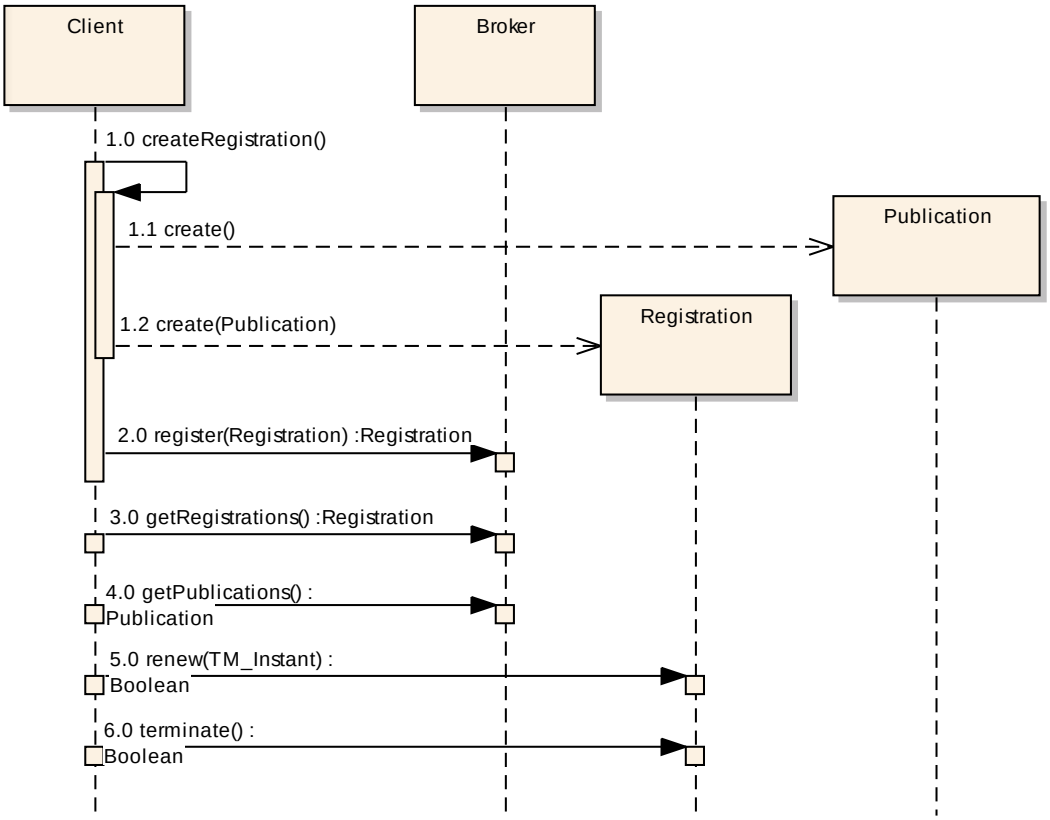

Figure 20: Creating a Registration

The interactions shown in Figure 20 are explained in Table 29.

Table 29: Discussion of Registration Creation Interactions

Message(s)	Notes
Msg: 1.0 *From:* *Client* *To:* *Client*	Creating a registration involves two steps: creating a publication and a registration proposal.
Msg: 1.1 *From:* *Client* *To:* *Publication*	The first step is to create the publication which contains information about the events that will be published as part of the registration.
Msg: 1.2 *From:* *Client* *To:* *Registration*	The second step is to create a Registration that can later be proposed to the Broker. Note: for simplicity only one Registration is modelled here. In a web service environment, it is more likely that two distinct registration resources will be created: one governed by the client which is proposed to the broker and one governed by the broker that was created based upon the proposed registration.
Msg: 2.0 *From:* *Client* *To:* *Broker*	The client registers at the Broker by sending a registration proposal. The Broker sends (a pointer to) the accepted registration in the response.
Msg: 3.0 *From:* *Client* *To:* *Broker*	The client may retrieve the list of all currently existing Registrations from the Broker.
Msg: 4.0 *From:* *Client* *To:* *Broker*	The client may retrieve the list of all currently existing Publications from the Broker.
Msg: 5.0 *From:* *Client* *To:* *Registration*	The client renews the registration by setting a new termination time.
Msg: 6.0 *From:* *Client* *To:* *Registration*	The client terminates the registration.

6.2.2.7 Demand Based Publishing

In deployments where an Event Service is a Broker it can be beneficial to pause publishers in case that no subscription exists at the Broker that is interested in notifications from that publisher. This mechanism is known as *demand based publishing*. The following behavior diagrams describe it in more detail.

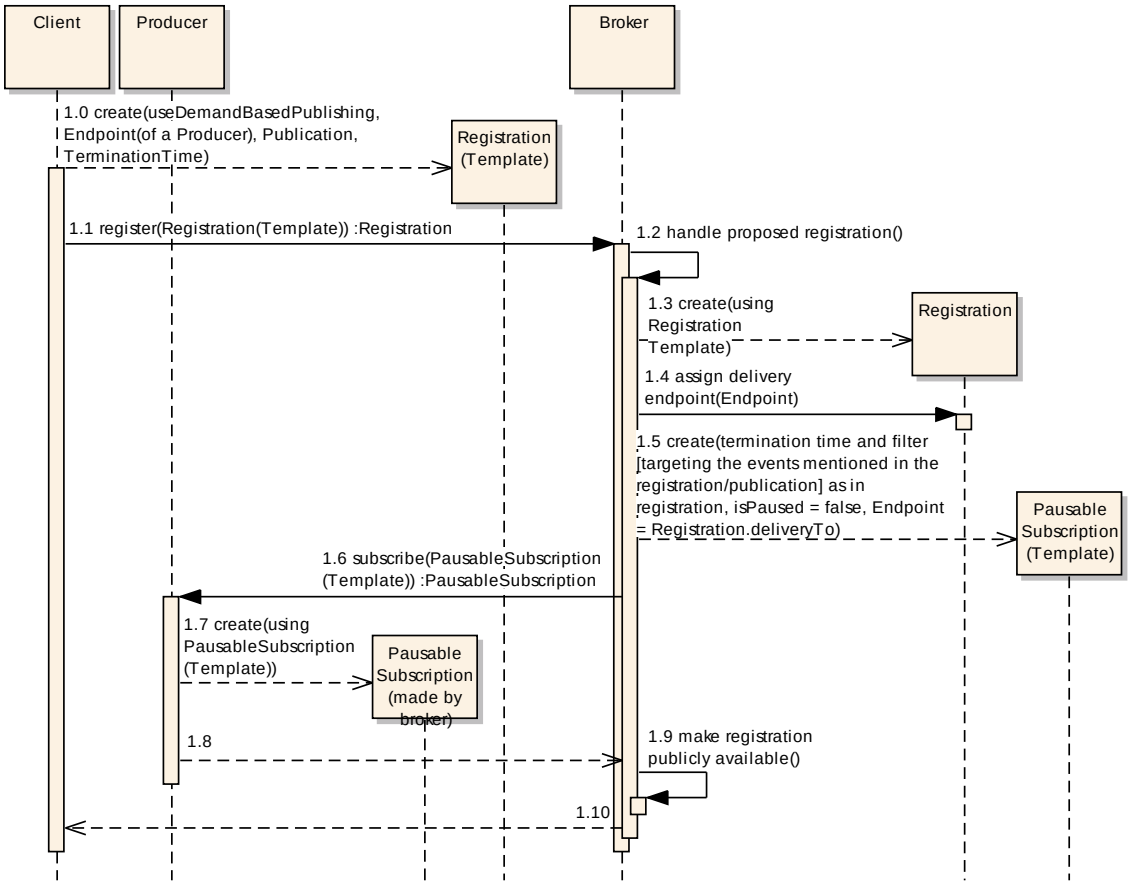

Figure 21: Registering a Demand Based Publisher

The diagram shows how a registration of a demand based publisher works. The interactions are discussed in the following table.

Table 30: Discussion of Interactions for Registering a Demand Based Publisher

Message(s)	**Notes**
Msg: 1.0 *From: Client* *To: Registration (Template)*	The client creates a Registration template. This template shall contain the policy indication requesting demand based publishing from the Broker. In addition, it shall contain the Endpoint of a Producer. The Registration template shall also contain a Publication with information which events are published by the Producer on which of its channels. This is important information for the Broker which needs to create a PausableSubscription targeting these events. A termination time for the Registration may be proposed by the

	client as well.
Msg: 1.1 *From: Client* *To: Broker*	The client registers the Producer as a new Publisher at the Broker. The previously created Registration template is send as part of the operation invocation.
Msg: 1.2 *From: Broker* *To: Broker*	The Broker has to perform several steps in handling the proposed Registration.
Msg: 1.3 *From: Broker* *To: Registration*	The Broker creates an internal Registration using the properties of the Registration template. The Publication associated with this new Registration should associate the published events with at least the same non-adhoc EventChannels as used by the Producer. It may associate the events with additional channels that are used by the Broker (both ad-hoc and normal channels).
Msg: 1.4 *From: Broker* *To: Registration*	The Broker assigns a delivery Endpoint to the Registration.
Msg: 1.5 *From: Broker* *To: Pausable Subscription (Template)*	The Broker creates a PausableSubscription template. This template is going to be proposed to the Producer. The Broker should ensure that the PausableSubscription has the same termination time as the Registration. This may be done by trying to set the termination time in the proposed PausableSubscription. If the Producer later on is rejecting the subscription for any reason related to the termination time then the Broker may apply an internal mechanism to ensure that the subscription created at the Producer is terminated (using the terminate() operation) or ignored (e.g. by shutting down the deliveryTo endpoint used in the subscription - explained later on). In addition, the Broker shall include a filter statement that targets the exact same events as advertised by the given Registration template. How the Broker creates such a filter statement is up to the implementation. Note: further work on filter languages is required to define best practices for this functionality. The subscription should not be paused initially. The mechanism of the Broker which checks for demand based publications that can be paused (see according diagram TBD) should be responsible for pausing and unpausing a PausableSubscription

	associated with a demand based publication. Furthermore, the Broker should use the delivery Endpoint of the new Registration as the delivery Endpoint of the PausableSubscription so that all events sent by the Producer can be associated with this Registration and the according Publication.
Msg: 1.6 *From:* *Broker* *To:* *Producer*	The Broker subscribes at the Producer (the Endpoint of which was given in the proposed Registration). The previously created PausableSubscription template is send as part of the operation invocation.
Msg: 1.7 *From:* *Producer* *To:* *Pausable Subscription (made by broker)*	The Producer creates an internal PausableSubscription using the properties of the PausableSubscription template.
Msg: 1.8 *From:* *Producer* *To:* *Broker*	(A pointer to) The new PausableSubscription is returned to the Broker.
Msg: 1.9 *From:* *Broker* *To:* *Broker*	The Broker, having successfully created a PausableSubscription at the Producer and having fulfilled the obligations in creating a demand based publication, makes the new Registration publicly available by adding it to its service metadata.
Msg: 1.10 *From:* *Broker* *To:* *Client*	(A pointer to) The new Registration is returned to the client.

Figure 22 shows how a *demand based publication* works, i.e. when the *PausableSubscription* associated with the publication shall be paused and resumed by a *Broker*.

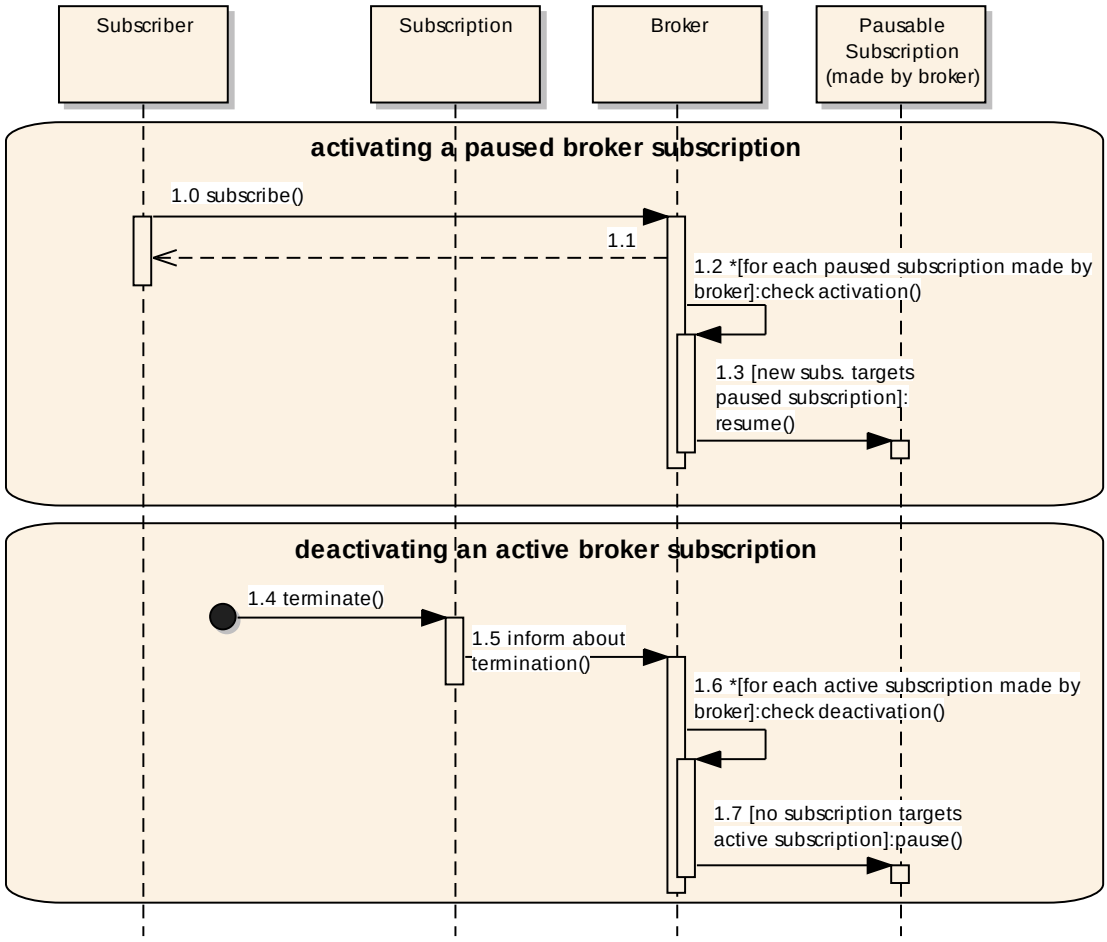

Figure 22: Handling a Demand Based Publication

The interactions shown in Figure 22 are discussed in the following table.

Table 31: Discussion of Interactions for Handling a Demand Based Publication

Message(s)	**Notes**
Msg: 1.0 *From: Subscriber* *To: Broker*	A Subscriber subscribes at the Broker.
Msg: 1.1 *From: Broker* *To: Subscriber*	The Broker creates the Subscription (not explicitly shown in diagram) and returns it to the Subscriber.
Msg: 1.2 *From: Broker* *To: Broker*	As a new Subscription was created, the Broker checks for each of the PausableSubscriptions that it created for a demand based publication - if that PausableSubscription is currently paused - if it needs to be reactivated / resumed.
Msg: 1.3 *From: Broker* *To: Pausable Subscription (made by broker)*	If the new Subscription targets events generated from a PausableSubscription that is currently paused, the Broker resumes that PausableSubscription.
Msg: 1.4 *From: <none>* *To: Subscription*	The terminate operation is invoked (from some system entity, e.g. a component that monitors the termination time of all resources with lifetime managed by the Broker) on a Subscription that is managed by the Broker.
Msg: 1.5 *From: Subscription* *To: Broker*	The Broker is made aware of the termination. How this is achived is implementation specific.
Msg: 1.6 *From: Broker* *To: Broker*	As a Subscription that was managed by the Broker is terminated, the Broker checks for each of the PausableSubscriptions that it created for a demand based publication - if that PausableSubscription is currently active - if it should be deactivated / paused.
Msg: 1.7 *From: Broker* *To: Pausable Subscription (made by broker)*	If the terminated Subscription targeted events generated from a PausableSubscription that is currently active and if no other Subscription (that is managed by the Broker) targets these events, the Broker pauses that PausableSubscription.

The following diagram shows the interactions that follow a renewal / termination of a Registration and the *demand based publication* that belongs to it.

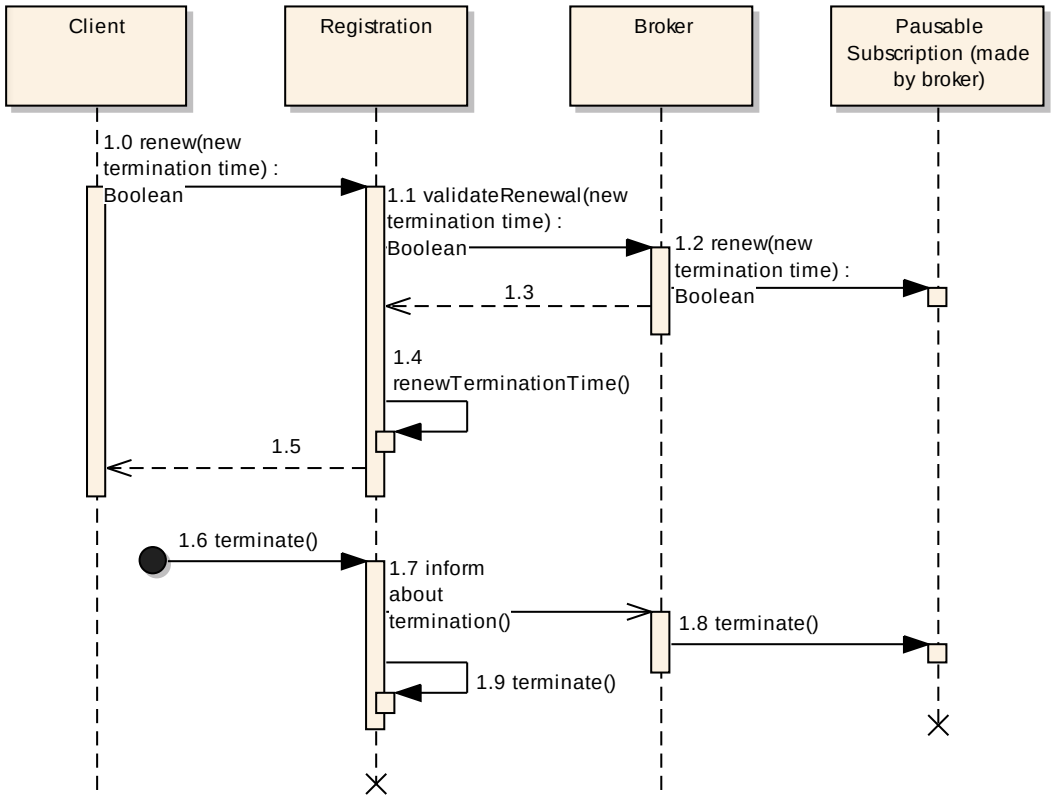

Figure 23: Renewing and Terminating a Demand Based Publication

The interactions shown in Figure 23 are discussed in the following table.

Table 32: Discussion of Interactions for Renewing and Terminating a Demand Based Publication

Message(s)	**Notes**
Msg: 1.0 *From:* *Client* *To:* *Registration*	A client renews the termination time of the Registration.
Msg: 1.1 *From:* *Registration* *To:* *Broker*	The Registration checks with the Broker if the termination time may be renewed to the proposed value. Note: How this callback is achived is implementation specific.
Msg: 1.2 *From:* *Broker* *To:* *Pausable Subscription (made by broker)*	The Broker tries to renew the termination time of the PausableSubscription that is associated with the demand based publication that belongs to the Registration.
Msg: 1.3 *From:* *Broker* *To:* *Registration*	In case that the PausableSubscription was renewed to the proposed termination time, the Registration may be renewed to that time as well, otherwise not. The former is assumed in the following interactions.
Msg: 1.4 *From:* *Registration* *To:* *Registration*	As the Broker agreed with the renewal, the termination time is set to the new value.
Msg: 1.5 *From:* *Registration* *To:* *Client*	The positive response to the renewal request is returned.
Msg: 1.6 *From:* *<none>* *To:* *Registration*	The terminate operation is invoked (from some system entity, e.g. a component that monitors the termination time of all resources with lifetime managed by the Broker) on a Registration that is managed by the Broker.
Msg: 1.7 *From:* *Registration* *To:* *Broker*	The Broker is made aware of the termination. Note: How this is achived is implementation specific.
Msg: 1.8 *From:* *Broker* *To: Pausable Subscription (made by broker)*	The Broker terminates the PausableSubscription that is associated with the demand based publication that belongs to the Registration.
Msg: 1.9 *From:* *Registration* *To:* *Registration*	The Registration terminates.

6.2.2.8 Ad Hoc Event Channels

Often multiple subscribers are interested in the same events. In support of sharing and reusing subscription filter criteria, an ad hoc event channel can be used. Such a channel is the result of a subscription that explicitly allows that all events matching the subscriptions filter criteria be also published on a channel that is created (by the Producer) ad hoc for that subscription. The identity of the subscriber that created the ad hoc channel can but does not have to be known. Other subscribers can investigate the description of the ad hoc channel and may subscribe for the events on that channel.

This is beneficial for subscribers that do not need to re-design a complex filter statement and also for an Event Service that does not need to perform complex filtering multiple times for multiple subscriptions that in fact are targeting the same set of notifications. An event service simply delivers the events that it would normally send only to the consumer of one subscription to multiple consumers that were all subscribed to the same set of events. Having complex logic to determine whether or not two subscriptions target the same set of events is thus not necessary.

The following activity diagram shows the interactions and events involved in creating an ad hoc channel. The activities of the provider and subscriber are performed in parallel.

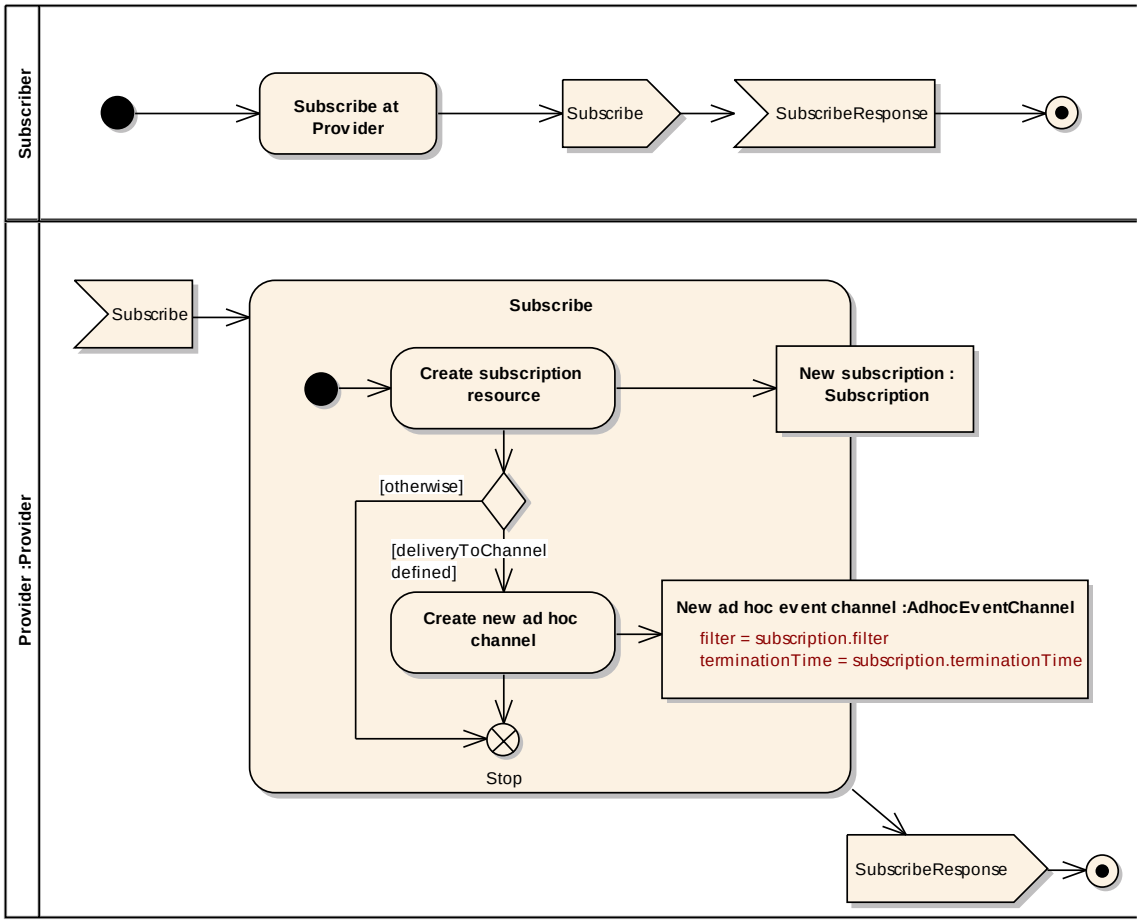

Figure 24: Creating an Ad Hoc Event Channel

The Subscriber sends the Subscribe event / message to the Provider.

The Provider performs the Subscribe activity upon receiving the Subscribe message. This involves creating a new subscription resource (object) and - if the deliveryToChannel property is defined in the Subscribe message - also creates a new AdHocChannel (object). That channel has the same filter and termination time as the new subscription. Upon completion of that activity the Provider emits a SubscribeResponse event / message which is received by the subscriber (keep in mind that the invocation of an operation may be performed synchronously but also asynchronously). This completes the activity of the subscriber.

6.3 Event Channels – A Concept to Facilitate Subscribing for and Filtering of Events

In an Event Architecture, clients consume events when they are published to them – such clients are called *event-driven*. The set of events available in an Event Architecture is sometimes called Event Cloud. Clients can tap into this cloud by subscribing to the events they are interested in. Subscriptions thus narrow down the set of events that needs to be sent to and consumed by a client. Different subscription models exist – OGC 09-032 section 8.1.3.1 provides an overview of some of these models.

Subscriptions based upon event channels leverage one particular model that will be discussed in the following. As described in OGC 09-032 section 8.1.3.1.1, an event channel can be compared with a TV channel – every news that is published on the channel is received by all the clients viewing that channel. The advantage of this approach is the performance gain for event services and clients.

In OWS-6 and OWS-7, content based filtering of events was primarily performed. The Event Service parsed each incoming event and processed it according to the filter statements of each existing subscription. Only events matching the filter criteria are published to the subscription's consumer endpoint, the other events are discarded – see Figure 25.

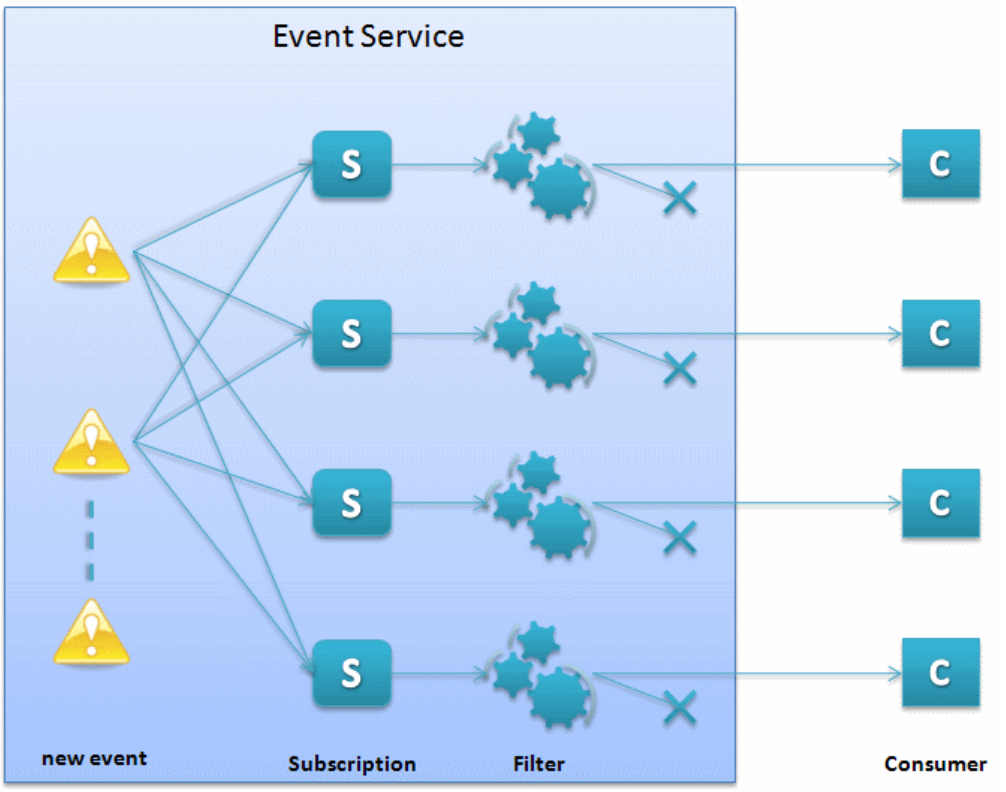

Figure 25: Event Service without event channels - subsetting of events only via content based filters

The more subscriptions exist (that need content filtering) the more processing time is required to handle each new event. An increasing amount of events can lead to an overflow of the event service, meaning that the service cannot process all incoming events due to a lack of processing capabilities. Increasing the amount of available processing resources through techniques like load balancing or cloud computing can solve the issue to a certain extent. A sophisticated Event Service may be able to apply more intelligent event matching techniques, for example by determining if the filter criteria of a new subscription are similar / comparable to an already existing one. Such techniques are not easy to realize, though.

This is where event channels can make a considerable improvement. They provide a way to partition the event cloud into smaller pieces. An application domain – for example the Sensor Web or Aviation domains – can define channels for the events relevant to them. Simple channels would be *sensor output* and *sensor status*. On the former, observations from sensors would be posted. All status events would be posted on the latter.

As we can see, events are assigned to channels based upon certain criteria. These are part of what we currently call channel semantics. A domain may define more criteria that need to be fulfilled by the events published on a channel. Constraints on spatial, temporal and thematic event properties are possible, among others. This allows us to create channels where only events are published that are forecasts for severe thunderstorms over the region of Florida within the next three hours.

Subscribers can simply look for event channels that publish the events they are interested in. This is facilitated if well-known channels defined by a certain domain are available. However, subscribers can also investigate the semantics of a given channel if according metadata is provided (for more information on event metadata and discovery, see section 6.2.1.3 and chapter 7). Once suitable channels were identified, subscriptions can be created against them – see the following figure.

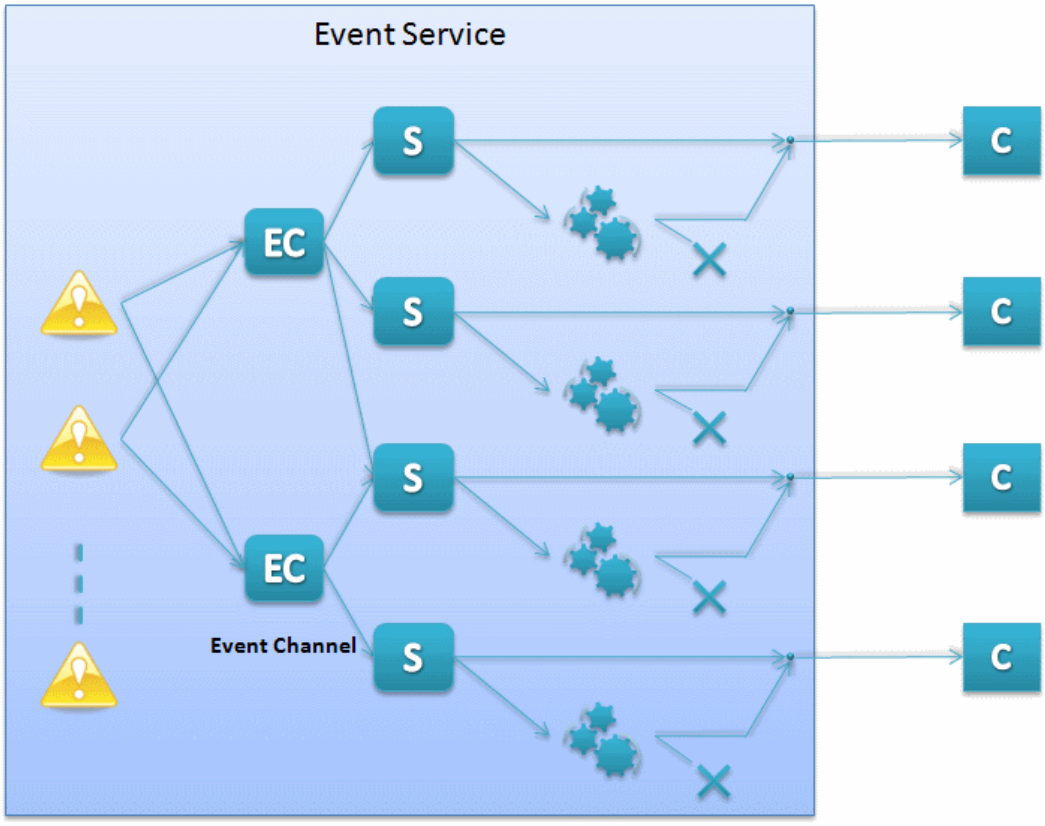

Figure 26: Event Service with event channels - subsetting of events by channel(s) and optional content based filters

The figure shows four subscriptions that receive events from two event channels. The Event Service only needs to assign incoming events to the appropriate channels once.

Each event assigned to a given channel can simply be forwarded to all the consumers that are subscribed to it. In Figure 26, three subscriptions are interested in the events of the above event channel. The more subscriptions exist that only use filter criteria to identify the channels that they want to receive events from, the more processing resources are saved that would otherwise be needed to perform content based event matching for these subscriptions. Of course, content based filtering can still be performed for subscriptions that listen to one or more event channels.

Some advanced channel concepts were identified during OWS-7: the aggregating and ad-hoc event channels. These are described in the following.

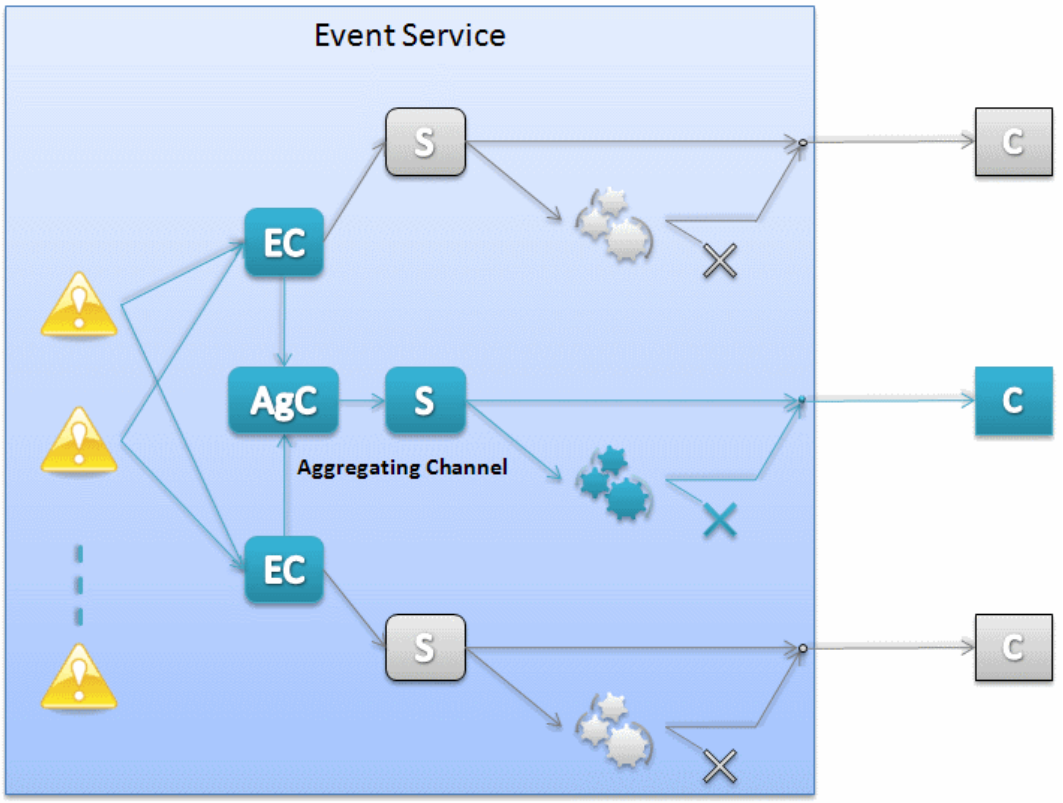

Figure 27: Aggregating Event Channels

A particular event can be published in different encodings on the same channel. As an illustration, think about a sports channel on TV. There, the opening ceremony of the Olympic Games is broadcasted (published) multiple times – one time in full length, another time as a short summary. The information provided on the ceremony in each event is quite different. We can say that the encoding of the event provides different information (compare with different feature types that capture very specific information in their properties).

Note: repeated publishing of the same event object in the same encoding on the same channel needs to be investigated in more detail – it may be a valid approach to publish an event multiple times but then such repetitions should be marked clearly as such.

An event may also be published on multiple channels. However, this is not necessarily the case. Therefore, if a client is interested in the events published on multiple channels, it has to create an according subscription. To support use cases in which some clients are interested only in specific events but other clients are interested in all of them, the concept of aggregating channels were defined. Remind the example where *sensor output* and *sensor status* events were published on two separate channels. An aggregating channel can combine the events from these two channels, allowing clients to subscribe to this one event channel more easily. A hierarchy of such aggregating channels is also possible. This can for example be of interest to model a system where local office get only specific events, regional offices get events from multiple local offices and the national office gets the events from multiple regional ones.

Often, clients are also interested in the same set of events that another subscriber receives. A channel publishing tornado warnings for a given area is one example. Although an Event Service provider may define a big set of event channels, he will most likely never be able to foresee each and every specific interest of his subscribers. Therefore, a subscriber may allow the creation of an ad hoc event channel that is tied to his subscription. Other subscribers may then create a subscription for that channel (which would need to get a meaningful description and metadata) and receive the same events that the original subscription does. The concept is depicted in the following figure.

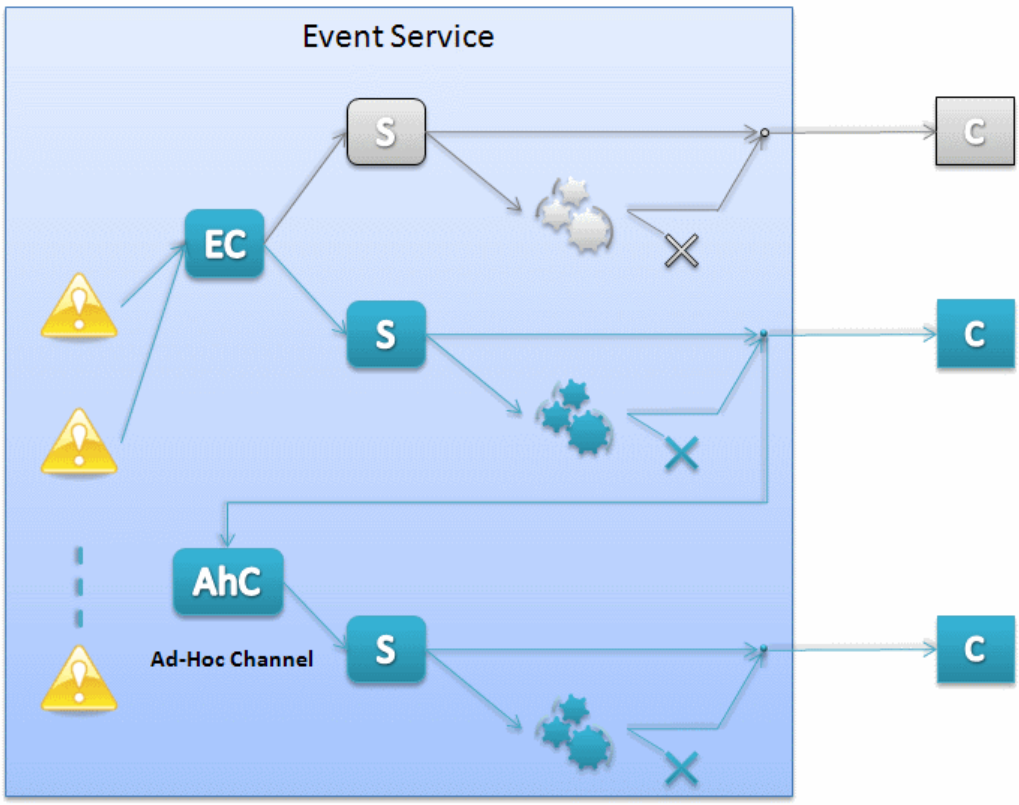

Figure 28: Ad-Hoc Event Channels

The figure shows two subscriptions for the same event channel (one in grey, the other in blue). The second subscription enabled an ad hoc event channel. As we can see, all events

– also those to which content based filtering is applied – are sent to both the consumer of the second subscription and the ad hoc channel. The third subscription is made against the ad hoc channel. Again, further content based filtering may be applied. This is performed at the discretion of the subscriber that created the third subscription.

Note: the concept of ad hoc event channels was tested in the Geosynchronization service work of the OWS-7 FDF thread (see section 11.1).

To summarize, this section described both the basic and advanced concepts of event channels. Event channels facilitate the arrangement of published events, event discovery (see chapter 7) and subscribing for events. Work in this area should be pursued, as it promises considerable performance gains for any Event Architecture deployment.

6.4 Publish / Subscribe Requirements

Section 6.2 documents publish / subscribe functionality that is relevant for OGC services in an abstract way. Both the overall information (documented using UML class diagrams) as well as behavior of a service is described.

A number of requirements were derived from the abstract publish / subscribe model. Figure 29 provides an overview of the requirements packages that were defined in OWS-7 together with their dependencies.

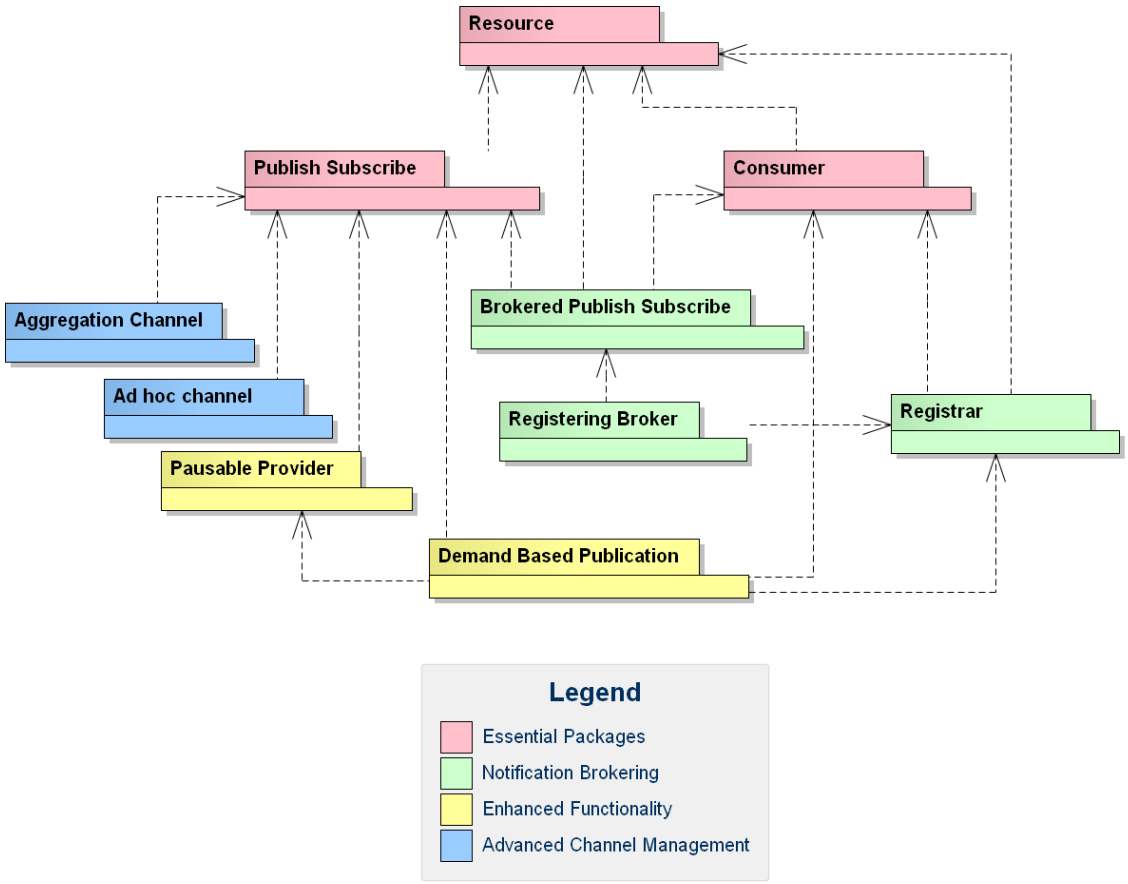

Figure 29: Event Service Requirements Packages

The packages were structured and colored according to their importance. Also, separation of functionality was aimed for.

The packages colored in read define the requirements that are essential for a publish / subscribe application:

- The *Resource* package defines the basic structure of resources, for example that a resource is identifiable and may have labels. In addition, resources with lifetime are defined. They can be terminated at a given time. Behavior to access resources and control their termination time is also defined.

- The *Consumer* package defines the structure of notifications as well as the behavior of system entities that receive them. The package supports the Consumer interface defined in OGC 09-032 section 6.6.2.1. It is defined separately to allow the inclusion in service roles that need Consumer functionality but not subscriptions (e.g. a Router as defined in OGC 09-032 section 6.6.1.5).

- The *Publish Subscribe* package defines structure of subscriptions and event channels, among others. In addition, behavior of a Provider, Publisher, Producer and subscriptions are defined. For a more detailed description of the according

interfaces and roles, see OGC 09-032 section 6.6. This package, together with the Resource and Consumer package, represents the core functionality of an Event Service.

The requirements packages that define functionality to enable notification brokering are colored in green:

- The *Registrar* package defines the structures and behavior to enable the registration of new publishers that send events / notifications to a Consumer. The package supports the Registrar interface defined in OGC 09-032 section 6.6.2.2. It is defined separately to allow the inclusion in service roles that need Registrar functionality but not subscriptions (e.g. a RegisteringRouter as defined in OGC 09-032 section 6.6.3.2).

- The *Brokered Publish Subscribe* package defines an extension of the functionality provided in the Publish Subscribe package. Behavior is defined that enables the implementation of a standalone Event Service that acts both as an event Consumer and Provider – see OGC 09-032 section 6.6.1.5. This setting was implemented in both OWS-6 and OWS-7.

- The *Registering Broker* package defines the requirements of a broker where new publishers have to be registered before they can send events to it. The according service role is described in OGC 09-032 section 6.6.3.3.

The yellow requirements packages define enhanced publish / subscribe functionality:

- The *Pausable Provider* package defines the structure of a pausable subscription and the behavior that needs to be supported for it.

- The *Demand Based Publication* package defines the requirements needed to support the mechanism described in more detail in section 6.2.2.7 of this report.

Finally, the requirements packages colored in blue define advanced event channel functionality:

- The *Aggregation Channel* package defines the structure and behavior needed to support aggregating event channels (see section 6.3).

- The *Ad Hoc Channel* package defines the structure and behavior needed to support ad hoc event channels (see section 6.3).

The details of the requirements contained in each package are provided in *Annex A - Publish Subscribe Requirements*.

6.5 Realization of Publish / Subscribe

6.5.1 Introduction

This chapter documents how well a given publish / subscribe technology fulfills the requirements derived from the abstract publish / subscribe model – see section 6.4.

6.5.2 Realization with WS-Notification

In October 2006 OASIS published WS-Notification[2] as an official standard. It is composed of three specifications: WS-BaseNotification, WS-BrokeredNotification and WS-Topics. The standard is for example used by the Sensor Event Service (OGC Discussion Paper 08-133) as the underlying technology to enable publish subscribe. The standard is primarily used in the SOAP binding but could be used in the Plain-old-XML (POX) binding as well, provided that a WS-Addressing binding for POX is defined.

The OWS-6 SWE Event Architecture ER (OGC 09-032) provides an Annex with a tutorial on how to use this technology. Section 6.5.2.2 complements that tutorial with additional information regarding the Notify message.

The following section discusses how well WS-Notification can realized the publish / subscribe requirements.

6.5.2.1 Requirements Mapping for WS-Notification

This section summarizes the results of the publish / subscribe requirements mapping study performed for WS-Notification. The detailed technical documentation of the requirements realization with WS-Notification is omitted here to avoid unnecessary distraction of the reader. It can be found in chapter 14 *Annex B – Publish Subscribe Requirements Realization Tables*, Table 53.

The following observations were made:

- The average realization grade is 2.15 (on a scale from 1 [perfect] to 4 [no go]), which indicates that WS-Notification in general at least partly already realizes the required functionality and that missing functionality can be added.

- There is no requirement with realization grade 4 (which would mean that the requirement cannot be realized with WS-Notification as underlying technology).

This shows that the WS-Notification standard is an option to implement publish / subscribe in OGC SOAP web service bindings.

[2] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

6.5.2.2 Considerations on the Notify Message sent via HTTP

The Notify operation is used to send notifications to consumers. Usually, no response is expected from the consumer upon receipt of this message. However if HTTP is used as underlying protocol each SOAP request sent via HTTP automatically has an HTTP response. The Web Services Interoperability Organization has some recommendations for SOAP-via-HTTP communication[3]. Especially section 3.8.3 on status codes is interesting as it defines the success status codes for HTTP responses.

As the HTTP specification [1] defines that a 200 (OK) response shall contain a message body while 202 (Accepted) should contain a message body and is intended to signal the acceptance of a process we recommend using 204 (No Content) as response code – this status code requires that no content is provided in the response body. This fulfills the semantics of the Notify response best and is within the recommendation of WS-I which states that *"An INSTANCE MUST use a 2xx HTTP status code on a response message that indicates the successful outcome of a HTTP request."* It also does not conflict with the HTTP defined semantics of the status codes. The response should not contain a body / payload.

Consequently, every publisher should be implemented to accept all 2xx codes as notify response.

NOTE: Usage of WS-ReliableMessaging results in at least some content communicated back to the entity that sent the notification in a reliable way.

6.5.3 Summary

Chapter 6.5 documents how well a given publish / subscribe technology supports the requirements derived from the abstract model. During the testbed, the participants were able to establish the mapping for one realization technology, namely WS-Notification – the results are documented in section 6.5.2.1. WS-Notification was already used in OWS-6 and was in OWS-7 also applied in the Aviation and SFE threads.

Other technologies, like Atom / AtomPub, WS-Eventing etc. can and should in the future also be investigated in this way. The foundation for that work is laid in this report.

When more technology mappings for the requirements have been achieved, a meaningful comparison of the different technologies can be made. That study will point out which technology is most appropriate for realizing publish / subscribe functionality required by OGC services per given OWS binding style (SOAP, RESTful etc).

[3] http://www.ws-i.org/Profiles/BasicProfile-2_0%28WGD%29.html#SOAPHTTP

7 Event Discovery

7.1 Introduction

An initial model for Event Service Discovery is presented in this chapter. A CSW-ebRIM registry extension package is developed to correspond with the Event Service Discovery model. Example Services are provided along with queries to demonstrate the discovery process. An online registry service is available to test the queries.

7.2 Event Metadata Example

The classes in the Event Metadata package described in section 6.2.1.3 were implemented in XML Schema to support testing of the event discovery functionality in OWS-7. The schema is provided in chapter 16, *Annex D – Event Metadata* .

Event metadata was created for two exemplary services, one from the Sensor Web and one from the Aviation domain. The metadata describes the event types that are published by the services on given event channels in certain encodings. Figure 30 and Figure 31 show the event metadata that was exemplarily created for OWS-7 in UML object diagram notation.

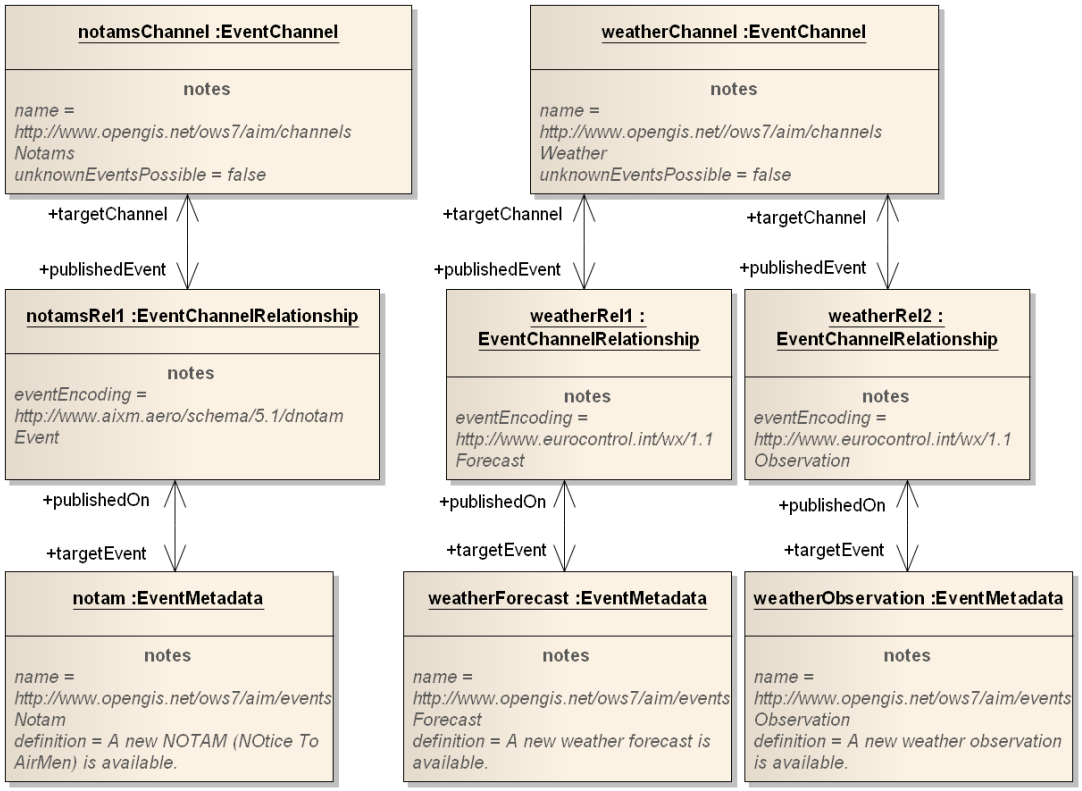

Figure 30: Example of Event Metadata for Aviation Service – UML Object Diagram Notation

The figure shows that three different event types are in use at the service. The events are published in AIXM and WXXM encodings (more specifically, as dnotam:Event and wx:Forecast / wx:Observation) on two event channels. Each channel has been assigned a specific name which uniquely identifies the channel semantics for a given domain.

Note: the XML implementation of the event metadata shown in Figure 30 can be found in *Annex D – Event Metadata XML Implementation*, section 16.2.

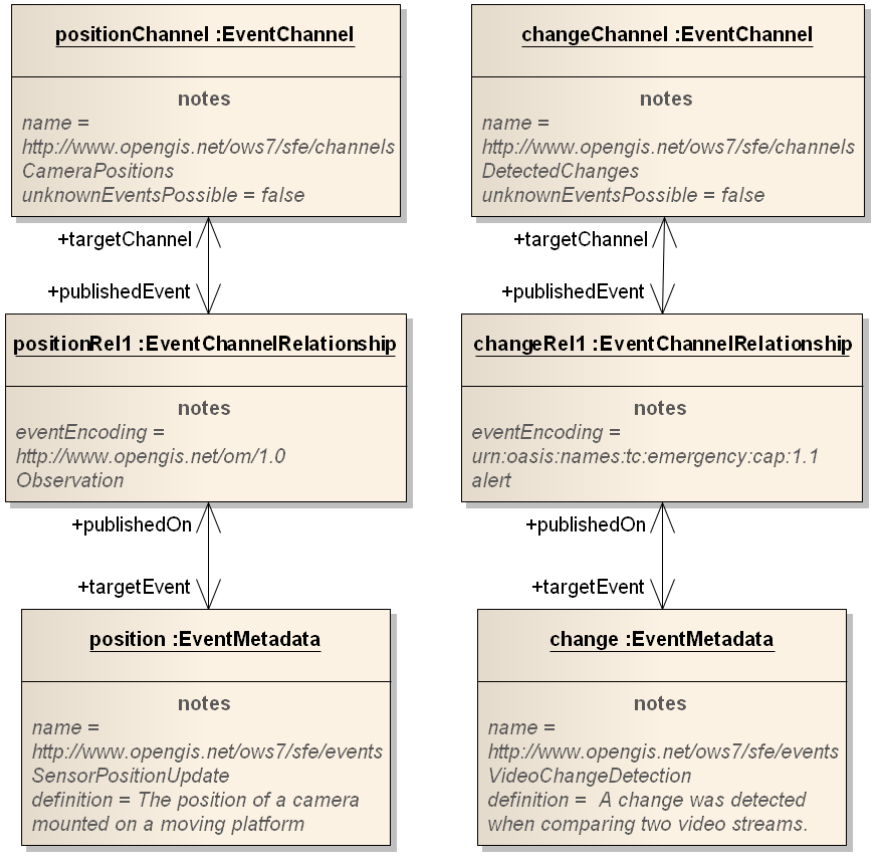

Figure 31: Example of Event Metadata for SFE Service – UML Object Diagram Notation

Figure 31 shows that two different event types are in use at the service. The events are published in O&M and CAP encodings on two event channels. Each channel has been assigned a specific name which uniquely identifies the channel semantics for a given domain.

Note: the XML implementation of the event metadata shown in Figure 31 can be found in *Annex D – Event Metadata XML Implementation*, section 16.3.

The example metadata described in above figures was used to test the event service discovery model, which is discussed in the following section(s).

7.3 The Event Service Discovery Model

The goal of the work described in this chapter is to be able to discover Event Services by querying event service metadata for both event channels and OGC Filter functionality.

Assumptions

1. The Event Services provide a method (i.e. GetCapabilities) from which one is able to obtain their OGC Capabilities document. This document lists the Services Filter Capabilities.

2. The Event Services provide a method from which one is able to obtain the TopicSet document which lists the Event Channels (e.g. "Topics"), Event Types and relationships.

3. A taxonomy representing each type of OGC Filter Capability defined in the OGC FE 2.0.0 is assumed to exist.

Querying based on Filter Capabilities can be accomplished by classifying each Service based on functionality declared in its Capabilities document and using the taxonomy that identifies the individual features to represent each function.

Querying based on a Service's supported Event Channels is accomplished by representing each channel and event-type as an object and then linking them together in a hierarchical manner.

```
Event Service  ----->  Event Channel [name="foo"] -----> Event-Type [name="bar"]
```

Figure 32 shows the model that is used for discovery. This model was created based upon the event metadata model described in section 6.2.1.3.

Note: *Annex C – Additional Event Discovery Material* - section 15.1 – contains the transaction request to populate a registry with this model.

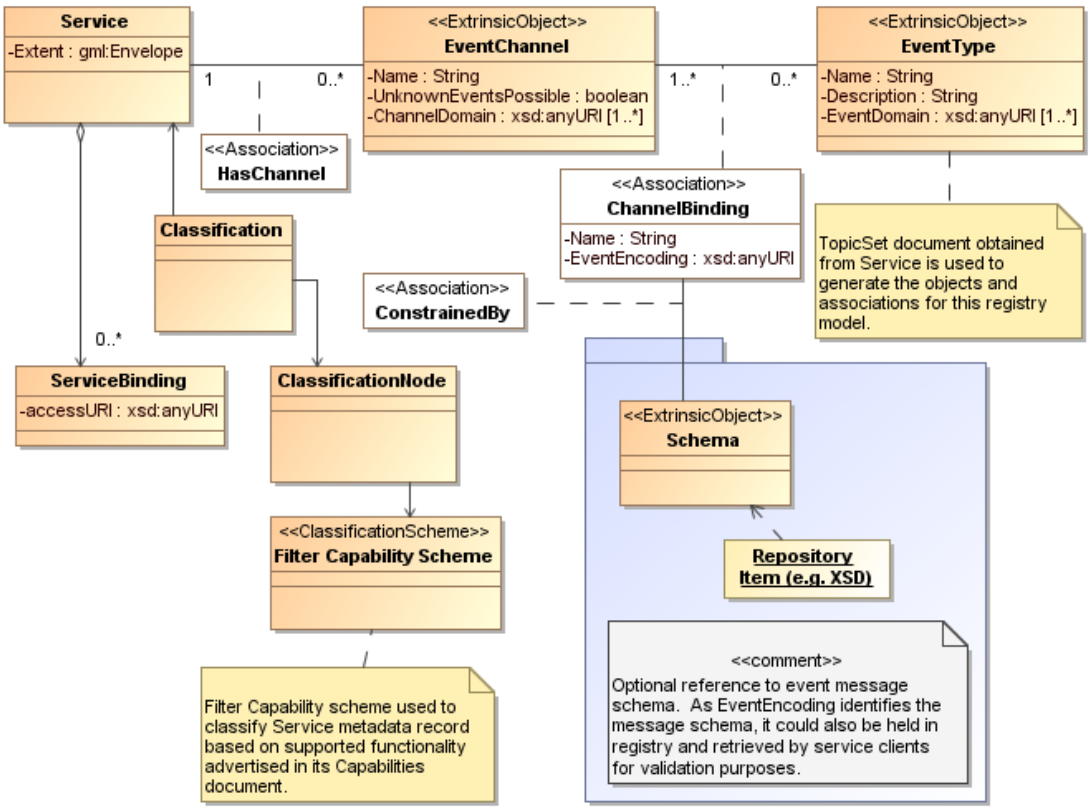

Figure 32: Registry Model for Event Service Discovery

7.4 The Event Service Discovery extension package

The components required to support the Event Service Discovery model are listed in the following table.

Table 33: Event Service Discovery Model Components

Component	Description of Component instances
Object Types	EventChannel, EventType
Association Types	HasChannel, ChannelBinding
Classification Node	Sensor event service type, AIM event service type (new C-Nodes added to existing ServiceType scheme)
Classification Schemes	Scheme for Filter 2.0.0 Operators and Operands

Note: The optional part of the model referring to a Schema object is not implemented in the current OWS project due to time constraints. It is documented in the model so it can be explored in a future OWS project.

The XML representation of the **Event Service Discovery extension package** is listed in section 15.1**.**

7.5 Event Serivce Discovery Scenarios

The following discovery scenarios are considered to be representative of typical search patterns that registry clients will follow in order to obtain a reference to a particular Service of interest. The query and expected response are provided in this document with links to a demonstration registry service. This is to enable the interested reader to exercise these scenarios in an interactive manner.

7.5.1 Scenario 1: Find a Service providing Sensor Data that supports Comparison Filters

In this scenario, a complex query is used to demonstrate the discovery of Services of interest based on both Filter capabilities and a specific Event type: Sensor events. The Event Discovery model is utilized as the associations between Services and EventChannels, as well as, between EventChannels and EventTypes, are traversed via multiple filter clauses.

The **EventDomain** property of an EventType is used to identify broad applicability to Sensor events (via URN **http://www.opengis.net/ows7/sfe/events**) which is combined with selection of a particular Sensor event type: **VideoChangeDetected**.

7.5.1.1 Step 1: Formulation of a CSW-ebRIM GetRecords query

A query is created using OGC Filter syntax that:

- selects Services which support Filter comparison operators.
- offers a channel that provides Sensor related events

Listing 1: Registry query to find services providing sensor events and supporting comparison filters

```
<?xml version="1.0" encoding="UTF-8"?>
<GetRecords xmlns="http://www.opengis.net/cat/csw/2.0.2"
  xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:wrs="http://www.opengis.net/cat/wrs/1.0"
  outputFormat="application/xml; charset=UTF-8"
  service="CSW" version="2.0.2"
  startPosition="1" maxRecords="10"
  resultType="results">
  <!--
    Find Services which provide Sensor related events
    -AND- which support the comparison operators
  -->
  <Query typeNames="Service Classification ExtrinsicObject_channel_event
Association_hc_cb">
    <ElementSetName typeNames="Service">full</ElementSetName>
    <Constraint version='1.1.0'>
      <ogc:Filter>
        <ogc:And>
          <!-- Service supports Filter Comparison operators -->
```



```
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>

<ogc:PropertyName>Classification/@classifiedObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>Classification/@classificationNode</ogc:PropertyName>
            <ogc:Literal>urn:x-ogc:def:rim-scheme:fes-2.0-
ops:comparison</ogc:Literal>
          </ogc:PropertyIsEqualTo>

          <!-- link Serivce to its Channels -->
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>
            <ogc:PropertyName>$hc/@sourceObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>$hc/@associationType</ogc:PropertyName>
            <ogc:Literal>urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasChannel</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>$hc/@targetObject</ogc:PropertyName>
            <ogc:PropertyName>$channel/@id</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>

          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>$channel/@objectType</ogc:PropertyName>
            <ogc:Literal>urn:ogc:def:rim-object-
type:ows7:EventChannel</ogc:Literal>
          </ogc:PropertyIsEqualTo>

          <!-- link Channel to its events -->
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>$channel/@id</ogc:PropertyName>
            <ogc:PropertyName>$cb/@sourceObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>$cb/@associationType</ogc:PropertyName>
            <ogc:Literal>urn:oasis:names:tc:ebxml-
regrep:AssociationType:ChannelBinding</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>$cb/@targetObject</ogc:PropertyName>
            <ogc:PropertyName>$event/@id</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>

          <!-- Event must be in SFE domain and handle some form of
"sensor measurement" -->
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>$event/Slot[@name="EventDomain"]/ValueList/Value</ogc:
PropertyName>

<ogc:Literal>http://www.opengis.net/ows7/sfe/events</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
```



```
<ogc:PropertyName>$event/Name/LocalizedString/@value</ogc:PropertyName>
            <ogc:Literal>VideoChangeDetected</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:And>
      </ogc:Filter>
    </Constraint>
  </Query>
</GetRecords>
```

7.5.1.2 Step 2: Query invocation

To invoke the query an HTTP-POST request is submitted to the registry query endpoint. Typically, a registry client is used to perform these operations.

For demonstration purposes, the Query utility page at http://registry.galdosinc.com/ows7/util/query can be used by pasting the above query into the text box and clicking the "Query" button.

Note: This Query utility page requires authentication --- use these credentials: ows7event / 0pen4me. The password contains two numbers, a zero (0) and a four (4).

Step 3: Extract the Service URL from the response. The base assumption is that the Service URL may then be invoked directly to access the Service in question. In reality, one would not expect to invoke such a service directly, but via a suitable client application.

The sample response for the query formulated above is shown below. The Service URL can be obtained from the Service object as indicated by this XPath statement:

```
/csw:GetRecordsResponse/csw:SearchResults/rim:Service/rim:ServiceBindin
g/@accessURI
```

assuming that the prefixes are bound appropriately. If the namespaces are not bound as expected, then this modified XPath should work:

```
/*:GetRecordsResponse/*:SearchResults/rim:Service/rim:ServiceBinding/@a
ccessURI
```

The following listing shows the response to the query shown in Listing 1. It has been simplified to focus only on relevant details.

Listing 2: Response from service registry listing a service that provides sensor events and supports comparison filters

```
<?xml version="1.0" encoding="UTF-8"?>
<GetRecordsResponse xmlns="http://www.opengis.net/cat/csw/2.0.2"
version="2.0">

  <SearchStatus timestamp="2010-05-27T13:53:14" />

  <SearchResults numberOfRecordsMatched="1" elementSet="full"
         numberOfRecordsReturned="1" nextRecord="0">
```



```
    <rim:Service xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
      xmlns:wrs="http://www.opengis.net/cat/wrs/1.0"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      id="urn:x-ows7:def:event-service:sfe:ex-01"
      objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:Service"
      status="urn:oasis:names:tc:ebxml-regrep:StatusType:Submitted">

      <rim:Slot name="http://purl.org/dc/elements/1.1/subject"
        slotType="urn:oasis:names:tc:ebxml-regrep:DataType:String">
        <rim:ValueList>
          <rim:Value>swe</rim:Value>
          <rim:Value>sfe</rim:Value>
          <rim:Value>sensor</rim:Value>
          <rim:Value>event</rim:Value>
          <rim:Value>ows</rim:Value>
          <rim:Value>ows7</rim:Value>
          <rim:Value>ows 7</rim:Value>
          <rim:Value>ows-7</rim:Value>
          <rim:Value>wrs</rim:Value>
          <rim:Value>ebrim</rim:Value>
          <rim:Value>ogc</rim:Value>
          <rim:Value>web service</rim:Value>
        </rim:ValueList>
      </rim:Slot>

      <rim:Slot name="Topics"
        slotType="urn:oasis:names:tc:ebxml-regrep:DataType:String">
        <rim:ValueList>
          <rim:Value>CameraPositions</rim:Value>
          <rim:Value>DetectedChanges</rim:Value>
        </rim:ValueList>
      </rim:Slot>

      <rim:Slot name="Extent"
        slotType="urn:ogc:def:dataType:ISO-19107:2003:GM_Envelope">
        <wrs:ValueList>
          <wrs:AnyValue>
            <gml:Envelope xmlns:gml="http://www.opengis.net/gml"
              srsName="urn:ogc:def:crs:EPSG::4326">
              <gml:lowerCorner>49.5964 2.3508</gml:lowerCorner>
              <gml:upperCorner>51.50167 6.4597</gml:upperCorner>
            </gml:Envelope>
          </wrs:AnyValue>
        </wrs:ValueList>
      </rim:Slot>

      <rim:Name>
        <rim:LocalizedString value="Sample SFE Event service #1" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value="Sample SFE Event service with two
event
          channels and full Filter 2.0.0 support." />
      </rim:Description>
          ...
```

```
      <rim:Classification id="urn:uuid:4ee124c5-3f72-43e5-ae43-
51a70d28770c"
        lid="urn:uuid:4ee124c5-3f72-43e5-ae43-51a70d28770c"
        objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:Classification"
        status="urn:oasis:names:tc:ebxml-regrep:StatusType:Submitted"
        classifiedObject="urn:x-ows7:def:event-service:sfe:ex-01"
        classificationNode="urn:ogc:serviceType:SensorEventService:0.3">
        <rim:Name />
        <rim:Description />
      </rim:Classification>
         ...
      <rim:Classification id="urn:uuid:5d8819d5-5484-4d58-946b-
e0f18fa4084d"
        lid="urn:uuid:5d8819d5-5484-4d58-946b-e0f18fa4084d"
        objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:Classification"
        status="urn:oasis:names:tc:ebxml-regrep:StatusType:Submitted"
        classifiedObject="urn:x-ows7:def:event-service:sfe:ex-01"
        classificationNode="urn:x-ogc:def:rim-scheme:fes-2.0-
ops:comparison">
        <rim:Name />
        <rim:Description />
      </rim:Classification>
         ...
      <rim:ServiceBinding id="urn:uuid:915a6135-92e2-44d2-8183-
588ef890d5fd"
        lid="urn:uuid:915a6135-92e2-44d2-8183-588ef890d5fd"
        objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ServiceBinding"
        status="urn:oasis:names:tc:ebxml-regrep:StatusType:Submitted"
        service="urn:x-ows7:def:event-service:sfe:ex-01"
        accessURI="http://www.foo-bar.net/events/sfe">
      </rim:ServiceBinding>
    </rim:Service>

  </SearchResults>
</GetRecordsResponse>
```

7.5.2 Scenario 2: Find a Service providing Aviation Data in a specified Area

In this scenario, a spatial query is used to demonstrate the discovery of Services of interest based on a defined region. This spatial clause is coupled with other OGC Filter clauses which filter based on an EventChannel offering Aviation data. The **ChannelDomain** property for an EventChannel is used to identify broad support for Aviation data.

7.5.2.1 Step 1: Formulation of a CSW-ebRIM GetRecords query

A query is created using OGC Filter syntax that:

- selects Services which supports an OGC Filter spatial operator (in this case BBOX).
- offers a channel that provides Aviation information and

- has data for the region around Vancouver, British Columbia, Canada.

Listing 3: Registry query to find services providing aviation events, supporting bbox filters and serving data for the Vancouver region

```
<?xml version="1.0" encoding="UTF-8"?>
<GetRecords xmlns="http://www.opengis.net/cat/csw/2.0.2"
  xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:wrs="http://www.opengis.net/cat/wrs/1.0"
  outputFormat="application/xml; charset=UTF-8"
  service="CSW" version="2.0.2"
  startPosition="1" maxRecords="10"
  resultType="results">
  <!--
    Find All Services which provide Aviation related events within
Europe
    -AND- which support the spatial operator BBOX
  -->
  <Query typeNames="Service Classification ExtrinsicObject Association">
    <?indicio-distinct-values true?>
    <ElementSetName typeNames="Service">full</ElementSetName>
    <Constraint version='1.1.0'>
      <ogc:Filter>
        <ogc:And>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>

<ogc:PropertyName>Classification/@classifiedObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>

<ogc:PropertyName>Association/@sourceObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>Association/@associationType</ogc:PropertyName>
            <ogc:Literal>urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasChannel</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>ExtrinsicObject/@id</ogc:PropertyName>

<ogc:PropertyName>Association/@targetObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>ExtrinsicObject/@objectType</ogc:PropertyName>
            <ogc:Literal>urn:ogc:def:rim-object-
type:ows7:EventChannel</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>ExtrinsicObject/Slot[@name="ChannelDomain"]/ValueList/
Value</ogc:PropertyName>
```



```
<ogc:Literal>http://www.opengis.net/ows7/aim/channels</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>Classification/@classificationNode</ogc:PropertyName>
            <ogc:Literal>urn:x-ogc:def:rim-scheme:fes-2.0-
ops:spatial:bbox</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:BBOX>

<ogc:PropertyName>Service/Slot[@name="Extent"]/wrs:ValueList/wrs:AnyValu
e</ogc:PropertyName>
            <!-- Vancouver, BC, Canada -->
            <gml:Envelope srsName='urn:ogc:def:crs:EPSG::4326'>
              <gml:lowerCorner>48.693 -123.832</gml:lowerCorner>
              <gml:upperCorner>50.051 -122.748</gml:upperCorner>
            </gml:Envelope>
          </ogc:BBOX>
        </ogc:And>
      </ogc:Filter>
    </Constraint>
  </Query>
</GetRecords>
```

7.5.2.2 Step 2: Query invocation

The invocation of the query is performed as described in section 7.5.1.2.

Listing 4: Response from service registry listing a service that provides aviation events, supports bbox filters and serves data for the Vancouver region

```
<?xml version="1.0" encoding="UTF-8"?>
<GetRecordsResponse xmlns="http://www.opengis.net/cat/csw/2.0.2"
version="2.0">

  <SearchStatus timestamp="2010-05-27T14:11:54" />

  <SearchResults numberOfRecordsMatched="1" elementSet="full"
    numberOfRecordsReturned="1" nextRecord="0">

    <rim:Service xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
      xmlns:wrs="http://www.opengis.net/cat/wrs/1.0"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      id="urn:x-ows7:def:event-service:aim:ex-01"
      lid="urn:x-ows7:def:event-service:aim:ex-01"
      objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:Service"
      status="urn:oasis:names:tc:ebxml-regrep:StatusType:Submitted">

      <rim:Slot name="http://purl.org/dc/elements/1.1/subject"
        slotType="urn:oasis:names:tc:ebxml-regrep:DataType:String">
        <rim:ValueList>
          <rim:Value>aim</rim:Value>
          <rim:Value>aixm</rim:Value>
          <rim:Value>event</rim:Value>
```



```
          <rim:Value>ows</rim:Value>
          <rim:Value>ows7</rim:Value>
          <rim:Value>ows 7</rim:Value>
          <rim:Value>ows-7</rim:Value>
          <rim:Value>wrs</rim:Value>
          <rim:Value>ebrim</rim:Value>
          <rim:Value>ogc</rim:Value>
          <rim:Value>web service</rim:Value>
        </rim:ValueList>
      </rim:Slot>

      <rim:Slot name="Topics"
        slotType="urn:oasis:names:tc:ebxml-regrep:DataType:String">
        <rim:ValueList>
          <rim:Value>notam</rim:Value>
          <rim:Value>weather</rim:Value>
        </rim:ValueList>
      </rim:Slot>

      <rim:Slot name="Extent"
        slotType="urn:ogc:def:dataType:ISO-19107:2003:GM_Envelope">
        <wrs:ValueList>
          <wrs:AnyValue>
            <gml:Envelope xmlns:gml="http://www.opengis.net/gml"
              srsName="urn:ogc:def:crs:EPSG::4326">
              <gml:lowerCorner>48.2244 -123.662</gml:lowerCorner>
              <gml:upperCorner>53.8783 -114.9169</gml:upperCorner>
            </gml:Envelope>
          </wrs:AnyValue>
        </wrs:ValueList>
      </rim:Slot>

      <rim:Name>
        <rim:LocalizedString value="Sample AIM Event service #1" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value="Sample AIM Event service with two
event
          channels and full Filter 2.0.0 support." />
      </rim:Description>
         ...
      <rim:Classification id="urn:uuid:cb92bb97-6edf-43e3-95fd-
6fb95adbfad6"
        lid="urn:uuid:cb92bb97-6edf-43e3-95fd-6fb95adbfad6"
        objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:Classification"
        status="urn:oasis:names:tc:ebxml-regrep:StatusType:Submitted"
        classifiedObject="urn:x-ows7:def:event-service:aim:ex-01"
        classificationNode="urn:x-ogc:def:rim-scheme:fes-2.0-
ops:spatial:bbox">
      </rim:Classification>
         ...
      <rim:Classification id="urn:uuid:e9c17a5f-a740-4872-9b29-
0280565adbee"
        lid="urn:uuid:e9c17a5f-a740-4872-9b29-0280565adbee"
        objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:Classification"
        status="urn:oasis:names:tc:ebxml-regrep:StatusType:Submitted"
```



```
        classifiedObject="urn:x-ows7:def:event-service:aim:ex-01"
        classificationNode="urn:ogc:serviceType:AimEventService:0.1">
      </rim:Classification>
         ...
      <rim:ServiceBinding id="urn:uuid:5f0a72e8-07c3-46ae-b4bd-
f5d3dcf6555b"
        lid="urn:uuid:5f0a72e8-07c3-46ae-b4bd-f5d3dcf6555b"
        objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ServiceBinding"
        status="urn:oasis:names:tc:ebxml-regrep:StatusType:Submitted"
        service="urn:x-ows7:def:event-service:aim:ex-01"
        accessURI="http://www.foo-bar.net/events/aim">
      </rim:ServiceBinding>
    </rim:Service>

  </SearchResults>
</GetRecordsResponse>
```

7.6 Sample Event Service objects

Two sample Service objects were used to construct and test the above scenario queries. These are also available in the OWS-7 demonstration registry as indicated by the links in the table below.

Table 34: Links to sample services used in the event service discovery scenarios

Service Description	Service Link
AIM Service ~ 2 event channels (Notam, Weather), full OGC Filter support	http://registry.galdosinc.com/ows7/query?request=GetRecordById&ElementSetName=full&view=urn:x-indicio:csw-ebrim:def:registry:transformation:RegistryBrowser&id=urn:x-ows7:def:event-service:aim:ex-01
SFE Service ~ 2 event channels (CameraPositions, DetectedChanges), full OGC Filter support	http://registry.galdosinc.com/ows7/query?request=GetRecordById&ElementSetName=full&view=urn:x-indicio:csw-ebrim:def:registry:transformation:RegistryBrowser&id=urn:x-ows7:def:event-service:sfe:ex-01

7.7 Discovery challenges:

In order to perform the scenarios illustrated here, clients must:

- construct an OGC Filter suitable for CSW-ebRIM services.
- understand the Event Service Discovery model well enough to formulate non-trivial queries.

Ideally, a suitable registry client would be available to hide many of the details visible in these discovery scenarios.

Specifically:

- it should not necessary to hand-craft XML queries to perform searches
- it should not be a requirement that one has extensive knowledge of a given registry model in order to effectively query against it.

8 Quality of service considerations

This section discusses Quality of Service aspects relevant in an Event Architecture. Emphasis is laid on security. Reliability is discussed shortly as well.

8.1 Event Security

8.1.1 Introduction

During OWS-7, initial work on Event (Service) Security was performed. The resulting discussions and findings are summarized in the following sections.

Note that the presented results represent the start of much more intensive investigation and testing that is needed to achieve a clear understanding of the relevant aspects and solutions for enabling security in an Event Architecture.

8.1.2 General Event Service Security Measures and Threats

The Event Service is an information broker in a SOA environment that routes messages from event sources to information consumers that have expressed interest in a particular type of information by subscribing to it. The Event Service sends the information to the subscribers on behalf of the original event sources. The messages can travel over multiple physical nodes in a network communicating via a mixed LAN and WAN infrastructure using different transport protocols (e.g. HTTP, JMS...).

8.1.2.1 Security threats and vulnerabilities

A vulnerability in security sense is any weakness that could be exploited to violate a system or the data it contains. A threat can be described as a potential violation of security.

The following threats might occur in the event service domain:

- Data destruction (e.g. subscriptions, publications, notifications…)
- Unauthorized creation of malicious subscriptions
- Unauthorized modification and corruption of data
- Theft and loss of data
- Unauthorized disclosure of data

- Service disruptions affecting SLA (Service Level Agreement)

Threats can be classified as accidental or intentional thefts:

- Accidental threats are threats that exist with no explicit intention (e.g. system errors and software defects).

- Intentional threats are threats which are explicitly planned and executed. They may range from using monitoring tools to very sophisticated security threats.

In addition, threats may be active or passive:

- Passive threats would not result in any modification to any data contained in the event system and where neither the operation nor the state of the system is changed (e.g. wire tapping).

- Active threats to a system involve the alteration of data changes or the state of the system (e.g. modification, deletion and creation of notifications).

Some specific threats and vulnerabilities are described in the following sections.

8.1.2.1.1 Denial of service

This attack may involve reducing traffic or it may generate extra traffic to the Event Service or broker. It is also possible to generate messages intended to disrupt the normal operations of the Event Service. For example, creation of too many subscriptions with complex filter may limit the processing power of an Event Service and as consequence critical subscriptions can no longer be handled properly or according to SLA.

8.1.2.1.2 Message replay

A message replay attack occurs when a (recorded valid) message, or any part of that message, is repeated to produce an unauthorized effect. For example, a valid message containing authentication information may be replayed by another entity in order to authenticate itself (as something that it is not) - spoofing identity. To detect and eliminate this attack, mechanisms should be used to identify replayed messages such as timestamps, nonces or message sequencing – these can also be used to prevent replay of messages.

8.1.2.1.3 Message modification

Modification of a message occurs when the content of a message or the whole message is altered without detection and results in an unauthorized effect. Even if a message is encrypted it might be possible to change it without even understanding it (e.g. malleability property of cryptographic algorithm). To enable the receiver to detect alteration, a digital signature can be applied by the sender to the message hash.

8.1.2.1.4 Spoof of identity

Spoof of identity is an attack where an entity pretends to be a different entity. This attack is usually used with some other forms of attacks. For example, an authorized entity with few privileges may use a false identity to obtain extra privileges by impersonating an entity that has those privileges.

8.1.2.1.5 Intermediary network devices

The proxy devices, firewalls, NAT (network address translators) are devices that act as intermediaries for data flow between various entities. These devices may prevent incoming data flow to be passed to its final destination or other intermediaries in case of a "multihop" system (e.g. data passes thought multiple devices or hops until it reaches its destination). This situation happens but is not limited to delivering notifications to subscribers. For example, a firewall might be configured to only accept a data flow as a response to a request originated from a domain protected by the device itself. NAT device might hide private IP addresses from being exposed into public IP domain. In addition, these problems might be aggravated by frequent usage of DHCP service (dynamic IP addressing) which allows IP address to be leased from a pool of IP addresses over certain amount of time or per session basis, DNS service (i.e. resolving URL symbolic address to actual IP address) and any combination of devices, services mentioned above or network topology. This kind of problems could be mitigated by configuring those devices to allow certain types of traffics and/or using static IP addresses. For example, NAT static address mapping entry could be added to allow incoming data flow from an entity (e.g. Event Service) which has a fixed public IP address to be translated to a private IP address of the destination entity (e.g. subscriber). A firewall could be configured to allow incoming traffic from entities using fixed IP addresses to be propagated to the final destination. Some protocols could be better designed to handle firewall problems transparently (e.g. SOAP over HTTP) while some people may argue that is not a good practice.

8.1.2.1.6 Trojan horse type of attacks

Trojan horse attack usually consists of authorized and unauthorized operations and is usually combined with other threats and attacks like unauthorized disclosure of data and message reply. For example, a subscriber may request notifications to be delivered to an entity that is not authorized to receive them.

8.1.3 Security measures

Security features usually increases the system costs. The cost could be reflected in system performance degradation and in increased complexity for implementing, configuring and maintaining the system.

Each security threat and vulnerability should be analyzed to determine whether or not an appropriate security measure is required. In addition, some threats might not be realized in practice due to system deployment environment (e.g. network topology) and trade-off between effort and risks that a particular threat will be exploited.

There are eight security measures identified to address particular aspects of security threats. They are discussed in the following sections.

8.1.3.1 Authentication

The authentication measure serves to confirm the identities of communicating entities. The authentication ensures the validity of the claimed identities of the entities participating in Event Service communication (e.g. publishers, subscribers, brokers...).

One common requirement in such an environment is the authentication of the participants in the system that communicate with each other as the information that is exchanged must come from trusted sources. This requires that two communication entities have to authenticate with each other no matter in which topologic position in the system they are. This could be achieved by exchanging security tokens between the entities to prove their identity or using some kind of distributed identity management system (e.g. brokered trust relationships) to manage identities within a particular domain or cross domains. A security token is essentially a collection of claims. The most common security tokens are Username/Password combination, SAML tokens and X.509 Public Key Infrastructure (PKI) or any combination of them. The PKI provides a standard for strong authentication, based on public key certificates and certification authorities. SAML is the framework for communicating user authentication, authorization, and attribute information in a distributed Web Services environment. SAML allows entities to make assertions regarding the identity, and attributes of a subject to other entities.

The authentication can be on a transaction, session, temporal or some other basis. For example, in case of distributed identity management the entity might obtain security tokens and establish security context for certain limited time period or session duration. In the context of the Event Service the authentication process may be performed when a data provider registers a data source with the Event Service and/or with every data event that is send to the Event Service. Often, a subscriber needs to authenticate itself with the Event Service in order to be allowed access to certain events and operations. It is important to note that the authentication security measure not only addresses user authentication but also has to authenticate the individual services that participate in the Event Service notification distributed processing.

8.1.3.2 Authorization or access control

The authorization measure protects against unauthorized access to certain resources (e.g. data, operations). The authorization measures ensure that only authorized entities (e.g. personnel, device, software agent...) are allowed access to stored information, information flows and services.

The authorization requirement in a SOA environment supporting the Event Service determines who can do what on which service under certain conditions.

Authorization policies can be attached to the subject that wants to access a service (e.g. Role Based Access Control) or assigned to the target that the subject wants to access (e.g. Resource Based) or both. The implementation of the authorization security can be

imperative, that is coded in the service that controls the resource or declarative, that is defined in policies (e.g. XACML policies). In the Event Service scenario, supporting mission critical application domains like Aviation, authorization can govern the access to particular data publications/events and for the registration of a publication of restricted data only from an authority that owns the data. In other words, only authorized entities shall be able to invoke certain operations of the Event Service. For example, the X.509 Attribute Certificates and XACML policies mechanism allow setting user access privileges in a multi-vendor and multi-application environment. SAML and SAML profiles allows entities to make assertions regarding the attributes and entitlements of a subject to other entities in a distributed environment.

The OGC extension of XACML, GeoXACML, allows the definition of spatially constrained access control policies that can be used to prevent access to resources in a certain area from organizations that are not authorities in this area or region, for example. Authorization policies may be based on duration of certain operation invocations, time of operation and invocations.

Authorization mechanisms may be applied at either end of entities involved in communication and at any intermediate points if desirable (e.g. firewall).

8.1.3.3 Non-repudiation

The non-repudiation measure provides means for preventing an individual or entity from successfully denying to have performed a particular action related to data (e.g. events) by making available proof of various actions (e.g. proof of publishing event, proof of delivering event to interested parties by the event service, proof of event origin, proof of event ownership...). It also ensures the availability of evidence that can be presented to a third party.

Sometimes, it would be necessary to have some kind of notarization mechanism/service by the third party to provide a non-repudiation service. A trusted third party might provide notarization mechanism by recording data flows or operation invocations between entities (e.g. relay messages) in order to provide proof and resolve potential disputes. A protocol between entities and notarization services must be well defined and agreed by entities involved in communication.

For example, in the Event Service brokered case the data publishers would ask for and obtain non-repudiation services from the Event Service, transmit the data, and receive notice that the data has been received and acknowledged by the Event Service. The non-repudiation services assure both the Event Service and data publisher that data has been successfully transmitted. On the subscriber side, the Event Service would ask for and obtain a non-repudiation service from a subscriber service, transmit the data, and receive notice that the data has been received and acknowledged by the subscriber service (event consumer).

Usually, the non-repudiation mechanism is provided by using some kind of digital signature mechanisms, data integrity mechanism, secure event logging and/or notarization service.

8.1.3.4 Data confidentiality

The data confidentiality measure protects data from unauthorized viewing. The data confidentiality ensures that the data content cannot be understood by unauthorized entities. Encryption and access control policies are some of the methods often used to provide data confidentiality.

The encryption mechanism could be applied to the whole message or any pieces of message involved in communication between entities or storage of data.

The encryption algorithm usually involves providing a key (e.g. predefined key, key generated during session handshake) and an actual algorithm. Some of the well known and popular algorithms are:

- AES (Advanced Encryption Standard). Key sizes 128, 192 and 256 bits
- DES (Data Encryption Standard). Key sizes: 56 bits
- 3DES (Triple Data Encryption Standard). Key sizes: 56, 112 (2* 56), 168 (3*56) bits
- BlowFish Key sizes: 8-448 bits

The key management could be provided by agreeing on (pre-shared) keys or using well defined standards for managing cryptographic keys (e.g. PKI standard).

8.1.3.5 Data integrity

The data integrity measure ensures the accuracy of data. In addition, this measure ensures that unauthorized modification, deletion, creation, and replication can be detected.

For example, the process of checking the data integrity requires the sending entity (e.g. the Event Service) to add to data some digest information (e.g. hash code, CRC...) which is a function of the data itself. This information may itself be encrypted. The receiving entity (e.g. subscribers) generates corresponding information and compares it with the received information to determine whether the data has been modified.

XML digital signature technologies are some of the most common form of implementing this measure in the SOA environment. Using message sequence numbers and timestamps may additionally provide the detection of message duplication.

Some of the most popular algorithms to generate digest information are:

- MD5
- SHA (e.g. SHA-1, SHA-2...)

The key management could be provided by agreeing on (pre-shared) keys or using well defined standards for managing cryptographic keys (e.g. PKI standard).

8.1.3.6 Availability

The availability measure ensures that there is no denial of authorized access to various information and services due to events impacting the system. One can also consider disaster recovery mechanisms to be included in this security measure category.

Some of the most common techniques for availability measures are to have potential to add parallel and/or backup processing services (e.g. stand by and fail-over services and capabilities). This includes choosing adequate network design and options.

The authorization and authentication measures would also ensure that no disruptive operations are allowed.

8.1.3.7 Privacy

The privacy measure provides for the protection of sensitive data that might be derived from the observation of Event Service activities. The privacy measure may also provide the right of entities to determine what information related to them may be collected, stored and disclosed and by which entities and which entities may have access to that information.

Examples of this data might include Event Service subscriber's or publisher's geographic location or some other privacy related information.

8.1.3.8 Communication security

The communication security measure ensures that data flows only between the authorized entities. This will also guarantee that the data is not diverted and intercepted as it passes between these entities.

If messages are always exchanged between an Event Service and a client without an intermediary, different levels of secure communication may already be achieved using various network layer security protocols (e.g. IPSec, HTTPS, TLS/SSL, MPLS, VPN...). In situations where messages have to travel over multiple physical nodes over a public network it will be required to protect the message itself, not just between two endpoints. This is the case in a federated brokered Event Service environment. To protect a message from the event publisher to the final consumer, independent of the transport protocol, data confidentiality and data integrity measures must be applied to the message itself.

8.1.4 Mapping security measures to threats

The "Yes" in a table designate that a particular security threat is mitigated by a corresponding security measure.

It is usually a combination of security measures which can mitigate certain threats (e.g. both authentication and authorization).

Table 35 and Table 36 show in which way each of the security threats and vulnerabilities discussed in section 8.1.2.1 is mitigated by the security measures described in section 8.1.3. This way, solutions to enable event security can be discussed in a technology independent way. The sections 8.1.5 and 0 describe how the necessary security measures can be realized in WS-* and RESTful environments.

Table 35: Mapping security measures to threats – part one

Security Measure\Threat	Data destruction	Unauthorized creation of malicious subscriptions	Unauthorized modification and corruption of data
Authentication	**Yes** Verify the identity of the entity attempting to delete data.	**Yes** Ensure that only authenticated entities can create subscriptions. This is in particular important if subscriptions are possible using "on behalf of".	**Yes** Verify the identity of the entity attempting to modify data.
Authorization	**Yes** Ensure that only authorized entities can delete data.	**Yes** Ensure that only authorized entities can create subscriptions. This is in particular important if subscriptions are possible using "on behalf of".	**Yes** Ensure that only authorized entities can modify data.

Security Measure\Threat	Data destruction	Unauthorized creation of malicious subscriptions	Unauthorized modification and corruption of data
Non-repudiation	**Yes** Provide a proof that certain operations related to data deletion actually happened (e.g. secure log, notarization, signatures...).	**Yes** Provide a proof that certain operation related to subscription creation actually happened (e.g. secure log, notarization, signatures...).	**Yes** Provide a proof that certain operation related to data modification and corruption actually happened (e.g. secure log, notarization, signatures...).
Data confidentiality	**No** Encrypted data can be deleted.	**No** Encryption is not applicable here.	**No** Certain encryption algorithms allow modification of data without actually decrypting data.
Data integrity	**Yes** The Event Service can provide a proof that data or part of the data has been deleted (e.g. storage, processing and transmission of data) by applying certain cryptographic techniques.	**N/A** Integrity of subscriptions is ensured by authorization.	**Yes** The Event Service can provide a proof that data or part of the data has been modified (e.g. storage, processing and transmission of data) by applying certain cryptographic techniques.
Availability	**No**	**No**	**No**
Privacy	**No**	**No**	**No**

Security Measure\Threat	Data destruction	Unauthorized creation of malicious subscriptions	Unauthorized modification and corruption of data
Communication security	**Yes** Ensure data transported between endpoints is not deleted.	**Yes** Ensure data transported between endpoints is not modified.	**Yes** Ensure data transported between endpoints is not modified.

Table 36: Mapping security measures to threats – part two

Security Measure\Threat	Theft and loss of data	Unauthorized disclosure of data	Service disruptions affecting SLA
Authentication	**Yes** Verify the identity of the entity attempting to view data.	**Yes** Verify the identity of the entity attempting to receive data.	**Yes** Verify the identity of the entity attempting to perform certain operations affecting availability of service.
Authorization	**Yes** Ensure that only authorized entities can access data.	**Yes** Ensure that only authorized entities can receive data.	**Yes** Ensure that only authorized entities can perform certain operations affecting availability of service (e.g. creating a number of subscriptions with complex filter...).

Security Measure\Threat	Theft and loss of data	Unauthorized disclosure of data	Service disruptions affecting SLA
Non-repudiation	**Yes** Provide a proof that certain operation related to theft and loss of data actually happened (e.g. secure log, notarization, signatures).	**Yes** Provide a proof that certain operation related to access and disclosure of data actually happened. (e.g. secure log, notarization, signatures).	**Yes** Provide a proof that certain operation related to service interruption actually happened.
Data confidentiality	**Yes** Every entity in the system can protect data flow, data processing and data storage against unauthorized access or viewing by applying certain cryptographic techniques (e.g. encryption / decryption).	**Yes** Every entity in the system can protect data flow, data processing and data storage against unauthorized access or viewing by applying certain cryptographic techniques (e.g. encryption / decryption).	**No** Encrypted data does not affect availability.
Data integrity	**No** Signed data can be viewed.	**No** Signed data can be stolen and viewed.	**No** Signed data can be viewed by unauthorized third party.
Availability	**No**	**No**	**Yes** Ensure that access to the Event Service by authorized entities cannot be denied.

Security Measure\Threat	Theft and loss of data	Unauthorized disclosure of data	Service disruptions affecting SLA
Privacy	**No**	**Yes** Privacy measure may allow data publishers and subscribers to determine what information the Event Service may collect, store and disclose related to those entities. For example, the Event Service might not be able to collect IP address and geographic location of subscribers or statistics about various entities.	**No**
Communication security	**Yes** Ensure data transported between endpoints is not intercepted.	**Yes** Ensure data transported between endpoints is not diverted.	**No** Availability is not affected by using secure communication.

8.1.5 Threat Mitigation in WS-* Environment

This section explains which technologies can be used in a WS-* environment to realize the security measures discussed in section 8.1.3.

Table 37: Realization of security measures in a WS-* environment

Security measure	Realization
Authentication	The entities may include message signatures in messages as specified by WS-Security [4],[6],[7]. WS-Trust [5] defines secure token service for managing and distribution of security tokens in distributed environment. WS-Security provides the means to attach security tokes in the message. These security tokens can be username/password, SAML tokens [10], X.509 certificates...

Security measure	Realization
Authorization	WS-Security provides the means to attach various tokens in the message. WS-Trust defines secure token service for managing and distribution of security tokens which may include authorization attributes in distributed environment. For example, SAML 2.0 profile of XACML 2.0 [11] may provide security token with certain authorization obligations. One can also use X.509 attribute certificates and certificate authorities. In addition, WS-Federation [9] provides authorization claims between federated partners.
Non-repudiation	XML signatures and WS-ReliableMessaging [2] may provide non-repudiation service. WS-ReliableMessaging defines ordered delivery, duplicate elimination, and guaranteed receipt while XML signatures ensures integrity of those artifacts – also see section 8.2
Data confidentiality	Messages can be encrypted as specified by WS-Security. WS-Security allows encryption of any combination of body blocks, header blocks, and any of sub-structures by either a common symmetric key shared by the producer and the recipient or a symmetric key carried in the message in an encrypted form. For example, XML encryption can be used to encrypt XML contents.
Data integrity	Include message signatures in messages as specified by WS-Security. For example, XML signatures can be used.
Availability	WS-Trust and WS-Federation provides a mean of brokering identity management. This would ensure that only authenticated and authorized users can access certain information and perform operations.
Privacy	WS-Federation provides protection of the privacy claims across organizational boundaries. For example, WS-Federation defines a parameter to request a security token message that indicates which claims are requested to be protected. It provides a standard for confidential tokens, parameter confirmation, obtaining privacy statements
Communication security	WS-SecureConversation [8] defines security context establishment, sharing, and session key derivation. Example of using security transport protocol is SOAP over HTTPS.

8.1.6 Threat Mitigation in RESTful Environment

This section explains which technologies can be used in a RESTful environment to realize the security measures discussed in section 8.1.3.

The suggested approach for the security implementation within a REST service is either 2-way SSL or HTTP Basic Authentication over 1-way SSL. This will provide point-to-point or transport level security. There are no specific standards for implementing security in a RESTful web service, although there are common practices.

Table 38: Realization of security measures in a RESTful environment

Security measure	Realization
Authentication	In two-way SSL authentication (mutual authentication), the SSL client application verifies the identity of the SSL server application, and then the SSL server application verifies the identity of the SSL-client application. Two-way SSL authentication is also referred to as client authentication because the application acting as an SSL client presents its certificate to the SSL server after the SSL server authenticates itself to the SSL client. Either self-signed certificates or certificates released by a trusted CA such as VeriSign can be used. In any case it is important to check the certificate for validity, including validation of revocation lists and the trusted root certificates. In the context of web browser clients, the use of the SAML 2 Web Browser SSO Profile can be used to create a security context that is typically maintained in secure cookies (cookies that are only transmitted via HTTPS connections). Http Basic Authentication over one-way SSL enables the application operating as the SSL client to verify the identity of the application operating as the SSL server. The server can implement or use an existing identity management system API, such as SAML, OpenAuth or OAuth, to verify a user's identity. This would require the users of the service to have an account with the third party - the Identity Provider.
Authorization	In an SSL implementation, either one-way or two-way, the digital certificate can contain information about a user including their role. The web service could authorize access to data based on the user's role. Authorization could also be achieved via custom code implemented inside the service that would allow an administrator to configure access to individual users or a group of users. A developer could leverage existing third party identity management system APIs and data stores, such as an LDAP repository, to determine access controls on their service.

Security measure	Realization
Non-repudiation	In a two-way SSL authenticated system, if both sides use digital certificates generated by a trusted certificate authority (CA), such as VeriSign, to do their auditing of interactions, then both parties are protected. With Basic HTTP authentication with one-way SSL, only trusted audit at the service side is possible. This audit log needs to be confidential and integrity protected to both the client and the service party to ensure trustworthiness.
Data confidentiality	Confidentiality of the information being passed is the main purpose of the SSL and TLS protocols. SSL is a secure way of transferring information between two points on the internet using encryption. When data is being passed between the client and the server on a network, the SSL responses are encrypted so that the data cannot be deciphered by a third party and the data remains confidential.
Data integrity	When data is being passed between the client and the server on a network, SSL helps guarantee that the data will not be modified in transit by that third party.
Availability	Responsibility for availability is placed on the server. Availability is slightly addressed with SSL since secure communications prevent malicious users from having direct access to the system.
Privacy	SSL provides privacy. It is provided through encryption, which scrambles traffic so that it is incomprehensible to a potential eaves dropper.
Communication security	SSL technology allows web browsers and web servers to communicate over a secure connection. In this secure connection, the data is encrypted before being sent, and then is decrypted upon receipt and before processing. Both the browser and the server encrypt all traffic before sending any data.

8.2 Reliability

Reliable delivery of notifications is a feature that is required in mission critical systems. Usually, additional technology is put in place to realize it.

For SOAP, WS-ReliableMessaging [2] is such a technology. It enables the following delivery assurances: *AtLeastOnce*, *AtMostOnce*, *ExactlyOnce* and *InOrder*. Service providers may indicate which assurances they support using policy statements. Likewise, clients can request certain assurances using policies.

A more detailed discussion of reliable messaging technology for SOAP and RESTful bindings is provided in chapter 8.8.1 of OGC 09-032.

9 Event filtering

In this chapter, various aspects that are relevant for event filtering are discussed. The first section elaborates the pitfalls that exist when filtering at an Event Service. The second section discusses languages for filter / processing events. Finally, spatial filtering of events is investigated in more detail.

9.1 Potential Pitfalls

This section briefly discusses some of the general pitfalls that need to be avoided for doing useful filtering with an Event Service.

9.1.1 Boolean Result for Filter Statement

An Event Service matches events that it generates or that it receives from some publisher(s) against the existing subscriptions. The service needs to decide whether a given event matches a subscription's filter criteria or not. Therefore, the filter used in the subscription needs to evaluate to a boolean value.

A filter language needs to define very clearly how such a result is computed. For example, an XPath expression may have a set of XML nodes as result. If no further rules are defined how the Event Service should handle the result of the XPath expression, it will produce an exception. Further details on how XPath expressions should be handled are provided in section 9.2.2. Here, the following observation is important: in order for a filter language to be usable in an Event Service, either the language itself clearly defines how to achieve a boolean result out of a filter statement encoded in that language or the Event Service defines the required rules. In the case of XPath, the Event Service could for example apply an implicit boolean() function call to the results of an XPath expression – see section 9.2.2 for further details).

If a subscription does not use a filter language but some form of processing language to identify the events that are of interest, the requirement to have a boolean result does not apply. The processing language then defines the results that are sent to a subscription's consumer endpoint. The Event Pattern Markup Language is such a processing language (see section 9.2.3).

9.1.2 Event Wrapper

Another problem to perform filtering / processing at an Event Service may arise if the service is instructed to or automatically applies a wrapper format to an event before a filter is applied to it. If the service does not indicate the exact behavior in this situation, then subscription filters may not work as expected. In cases where the Event Service implementation uses a wrapper that is only a message container for multiple events, it should apply content based filtering on each of the events contained in the wrapper. The WS-Notification Notify message represents such a container.

9.1.3 Resolve Content Given By Reference

The GML model allow values of feature properties to be given by a reference (a URL in an xlink:href property) rather than inline of the XML instance. The idea is that a system that receives such an XML instance would automatically resolve the reference to get the value and insert it into the inline content.

An Event Service should automatically retrieve event properties given by reference if they are queried in the filter statement(s) of a subscription. This can happen on demand. Problems can occur if the value cannot be resolved (e.g. if wrong connection timeouts have been chosen) or if the value itself is complex and has properties given by reference. Such deep reference structures can slow down an Event Service.

The effects and possible performance issues of using event properties given by reference in event filtering should be investigated in more detail.

When using xlink references encoded according to the GML specification, data providers that create event instances should be aware that applications receiving the events may not preserve the namespace-prefix binding for the XPath expression used in the reference.

Consider the following example:

```
xlink:href="http://www.aixm.aero/schema/5.1/dnotam/example#
element(//dnotam:Event[gml:identifier='aPreviousId'])"
```

It shows an xlink with a URI reference that the GML standard (OGC 07-036) calls an *element() scheme based XPointer*. The absolute URI (http://www.aixm.aero/schema/5.1/dnotam/example) is followed by the fragment identifier and then the XPointer. Although this behavior is not recommended by the W3C, an application that receives an event (e.g. an XML router application) could change the prefixes in the attributes and elements of an XML instance but not in the content of them. In the above example, the *dnotam* and *gml* prefix would then no longer bind to the correct namespaces. A service could no longer resolve the correct element from the given URI.

Therefore, it is more safe to use an *xpointer() scheme based XPointer* (see OGC 07-036) – the example would then look like this:

```
xlink:href="http://www.aixm.aero/schema/5.1/dnotam/example#
xmlns(dnotam=http://www.aixm.aero/schema/5.1/dnotam)
xmlns(gml= http://www.opengis.net/gml/3.2)
xpointer(//dnotam:Event[gml:identifier='aPreviousId'])"
```

Here, the namespace-prefix binding is preserved in the reference itself.

9.1.4 Reference System Transformation

When a filter statement is created to be used in subscriptions, it often includes filter operand values. In simple applications, the reference system of these operands (e.g. a

geometry, a time or some thematic property) is the same as the one that will later be filtered upon. In other words, the filter operations can easily be performed. It is also possible to ignore the reference systems completely and to assume that they are the same. An example of an according filter statement would be *"air temperature is below 10"* and of an appropriate event *"observation air temperature is 8.7 °C"*.

However, in an Event Architecture this may not always be the case. Here, an Event Service can offer functionality to automatically transform the values of the filter operands and those of the incoming events to a compatible reference system so that the filter operation can be executed. The Sensor Event Service (OGC Discussion Paper 08-133) defines support for unit conversion. It does not define a generic mechanism to solve the issue. Further work to define such a mechanism would be beneficial, especially with respect to the heterogeneity of events and event sources in a cross-domain Event Architecture.

9.2 Filter / Processing Languages

9.2.1 Filter Encoding

The OGC Filter Encoding Specification (FES) supports filtering of generic XML instances. This is achieved via XPath expressions that are used to point to attributes or elements in the XML. These properties provide the values that a filter operates upon.

FES supports a wide range of filter functionality (temporal, spatial and thematic filters). It also supports additional functions which can be vendor specific or defined in standards.

Applications need to take care that the namespace-prefix binding used for XPath expressions of a filter statement are preserved – this is explained in more detail in the following section.

9.2.2 XPath

The XML Path Language (XPath[4]) supports filtering of XML instance documents. It provides simple comparison functions but also logical and temporal operations[5]. So far no spatial operations are supported. In addition, operations do not take into account the possible meaning of a complex XML element – like a gml:TimePeriod element contained in the XML instance.

Applications need to take care that the namespace-prefix binding used in an XPath expression are preserved and not destroyed by changing the namespace prefixes of XML instances. This can result in the expression no longer being able to identify the correct XML attributes and elements (in other words: the event properties).

---

[4] http://www.w3.org/TR/xpath20/

[5] http://www.w3.org/TR/xpath-functions/

One way to solve this issue could be to convert the XPath expression into a simple XQuery expression. For example, if the XPath expression would be:

```
//*[abc:PropertyX]
```

then it could easily be augmented by prepending a list of namespace declarations according to the XQuery specification. The resulting XQuery expression would then be:

```
declare namespace abc = "http://www.opengis.net/abc/1.0";
//*[abc:PropertyX]
```

This requires a service to support XQuery transformations. However, this mechanism could also become an OGC extension of XPath that would be well-documented in the appropriate documents.

As described in section 9.1.1, an XPath expression used to filter events needs to be evaluated by an Event Service to a boolean expression. The service can then unambiguously determine if a given event matches a subscription's filter criteria. When using XPath as a filter language in an Event Service, the service should apply an implicit boolean() function call to the results of an XPath expression – evaluated according to the XPath functions specification[6].

9.2.3 Event Pattern Markup Language

Filtering not only single events but whole event streams is what the Event Processing domain is concerned with. The order of events in the stream, their timing as well as patterns that can be detected are of interest here, among others. Detection of event patterns allow the derivation of higher level information from low level events. Critical information shall be produced as soon as possible in order to react to it.

Event processing nowadays is primarily concerned with financial applications. However, the techniques are more and more also used in other domains as well – also in the geospatial domain.

The *Event Pattern Markup Language* (EML; OGC 08-132) defines basic event processing functionality. The pattern language can be incorporated in processing languages like SensorML but may also extend the filter and processing functionality offered by OGC event services.

When using EML, one should keep in mind that it is currently an OGC Discussion Paper and not an approved OGC Standard. Therefore, the concepts described in the specification as well as the markup language to express event patterns may change significantly in the future (even though that may not be intended at this time). In addition, no standards have been defined in the event processing domain yet. In other words, there is no *SQL for Event Processing* – even though some vendors may claim that their

[6] http://www.w3.org/TR/xpath-functions/#func-boolean

products support such a thing. The EML was therefore developed based upon concepts that were identified to be common to multiple event processing languages.

EML users should consider the language as a testing ground to improve the filtering and processing functionality of standards based geospatial services and applications. The functionality provided is especially interesting for use cases where critical information needs to be derived from the data produced by various sources and clients be informed about new information as soon as possible. The Sensor Web domain is an example where Event Processing has already successfully been tested (see the *OGC Sensor Event Service* Discussion Paper, OGC 08-133). Other domains can benefit from the technology as well, for example the transportation domain.

9.3 Spatial Filtering of Events

9.3.1 Spatial Filtering via Bounding Box

Filtering published events based upon spatial filter criteria is a common use case. Especially in the geospatial domain events usually have a spatial context, in addition to their temporal context (the event time).

Events that are encoded as GML features share a common property: the *boundedBy* property of type GM_Envelope (see figure D.45 in OGC 07-036). The GML 3.2.1 schema currently allows the inclusion of a spatial or spatio-temporal envelope / bounding box of n dimensions as property value. In OWS-6 and OWS-7, the boundedBy property was used to perform spatial filtering of events. Several issues were identified with this approach. They are described in the following.

First of all, filtering events using the boundedBy property is in most cases inexact. It depends upon the accuracy required as well as the extent of the two geometries that are used for filtering whether this inexactness is relevant or not. Consider the following figure.

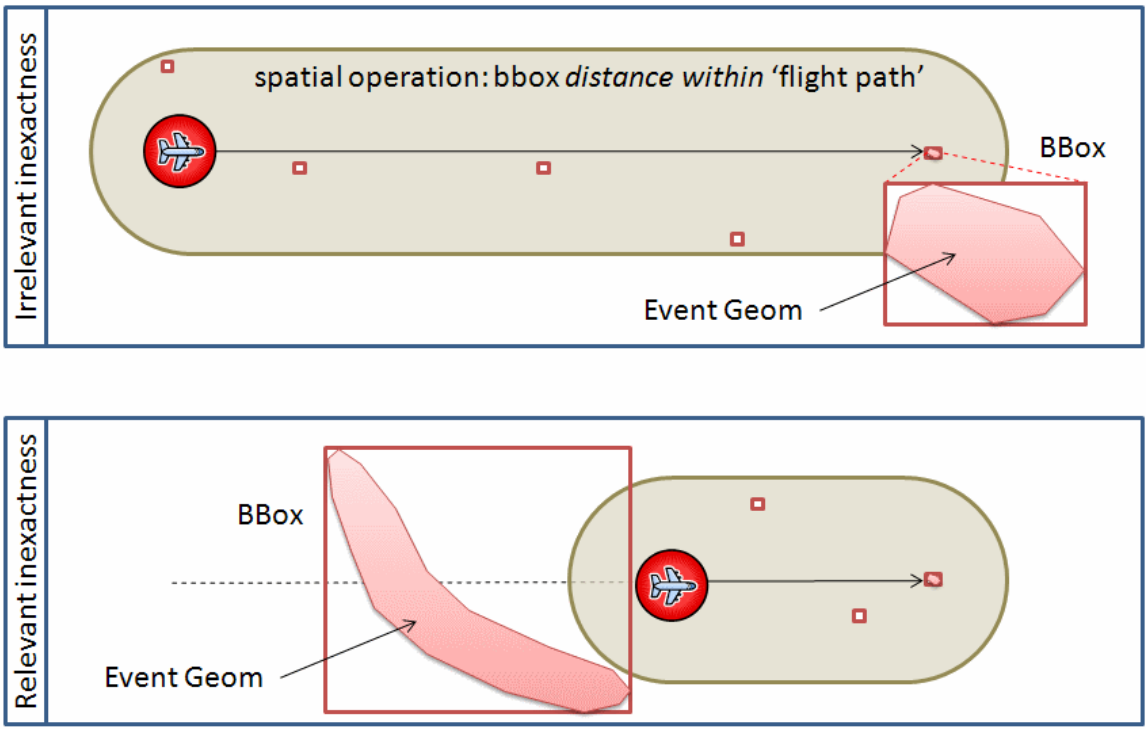

Figure 33: Spatial filtering using bounding boxes

The figure illustrates a use case in which all events that fall within a certain distance around the remaining flight path of an airplane are of interest. All other events shall be removed using a filter with the *dwithin* spatial operator. In the upper box, the inexactness introduced when filtering using the bounding box of an event's geometry is irrelevant because the geometry itself is very small compared to the buffer implied by the spatial operator. In the lower box, the inexactness becomes relevant. Even though the event itself – for example a weather observation or forecast (thunderstorm, volcanic ash cloud etc.) – is no longer of interest to the airplane, it will be reported to it. The event's geometry is not within distance to the remaining flight path – however, the bounding box computed from the event's geometry is. Apparently, it depends upon the use case and the geometries of the involved events whether the inexactness of using bounding boxes for filtering is significant or not.

One way to solve the issue of inexact boundary information would be to allow more detailed geometries in the boundedBy property of a GML feature. A comparable solution would be to add a common *spatialExtent* property to each event type used in a domain that is populated with the according geometry (the revised Event Model defines such a property - see section 12.2.4 for further information).

One thing to keep in mind is that spatial filtering using simple bounding boxes is usually faster compared to filtering where more complex geometries are used. This advantage may outweigh the issue when doing filtering using bounding boxes. However, additional factors may be important as well, for example to keep the amount of irrelevant data that is transmitted to a client as low as possible (e.g. important in networks with bandwidth limitations). The priority of the various factors depend upon the given use case.

Another issue using bounding boxes for filtering is that the property is optional in GML features. Currently, no rules or policies exist with which the behavior to populate the boundedBy property of a feature is governed. A data provider is free to populate the property for a given feature.

Note: when doing eventing some domains, for example the aviation domain, use events that contain information about multiple features – the event itself also being a feature. In that case, both the event itself and the contained features may have a spatial context from which bounding box information is computed and provided. Clients thus need to take care that they reference the correct boundedBy property in the filter statements they create.

To fully support spatial filtering upon boundedBy data, clients need to know if all the events with spatial context populate this property with correct data. At the moment a service does not indicate that this is supported. This may be doable using a service's metadata.

In addition, rules for populating the boundedBy property for a given feature type should be developed. Ideally, the according instructions are machine readable so that a service can discover them at runtime and compute the boundary of a feature on-the-fly. It would be good to investigate an according mechanism in the future, as it would benefit the filtering of events in multiple domains. Without such a mechanism, the boundary of an event would necessarily have to be provided by the service that generated the event in the first place.

9.3.2 Spatial Filtering of Events Using Dynamic Filter Properties

Usually, a filter statement that uses spatial operators is created by providing a fixed geometry as one component of the operation and a reference to a spatial event property as another component. For example, a filter to identify all events that are contained in a certain region would include the geometry of that region as one operand and a pointer to identify the correct geometry of the events as the other operand.

In some use cases, it would be beneficial to have the filter operands be updated dynamically. Think of the following example: a car navigator application computed a route and is notifying the driver about all events that affect the route – for example traffic jams, accidents, weather events, changed gas prices at gas stations along the route etc. A simple filter would create a buffer around the route. That buffer is computed once and all events that fall within the buffer are picked up by the application. However, as the car progresses on the route, the events that are located behind the car's current position are probably no longer relevant to it. Therefore, the buffer geometry with which events are filtered should dynamically be updated from time to time to account for the car's progress.

The use case described before can be mapped to the aviation domain. There, aeronautical and weather events that affect a planned flight (e.g., severe weather and airport closures) are of interest and should be reported to the pilot of the aircraft. All other events should not be reported to the pilot as they do not concern him. Again a buffer is computed around the route of the flight. If the buffer would change based upon the updated position of the aircraft, irrelevant events that are behind the aircraft would be ignored. This is shown in the following figure.

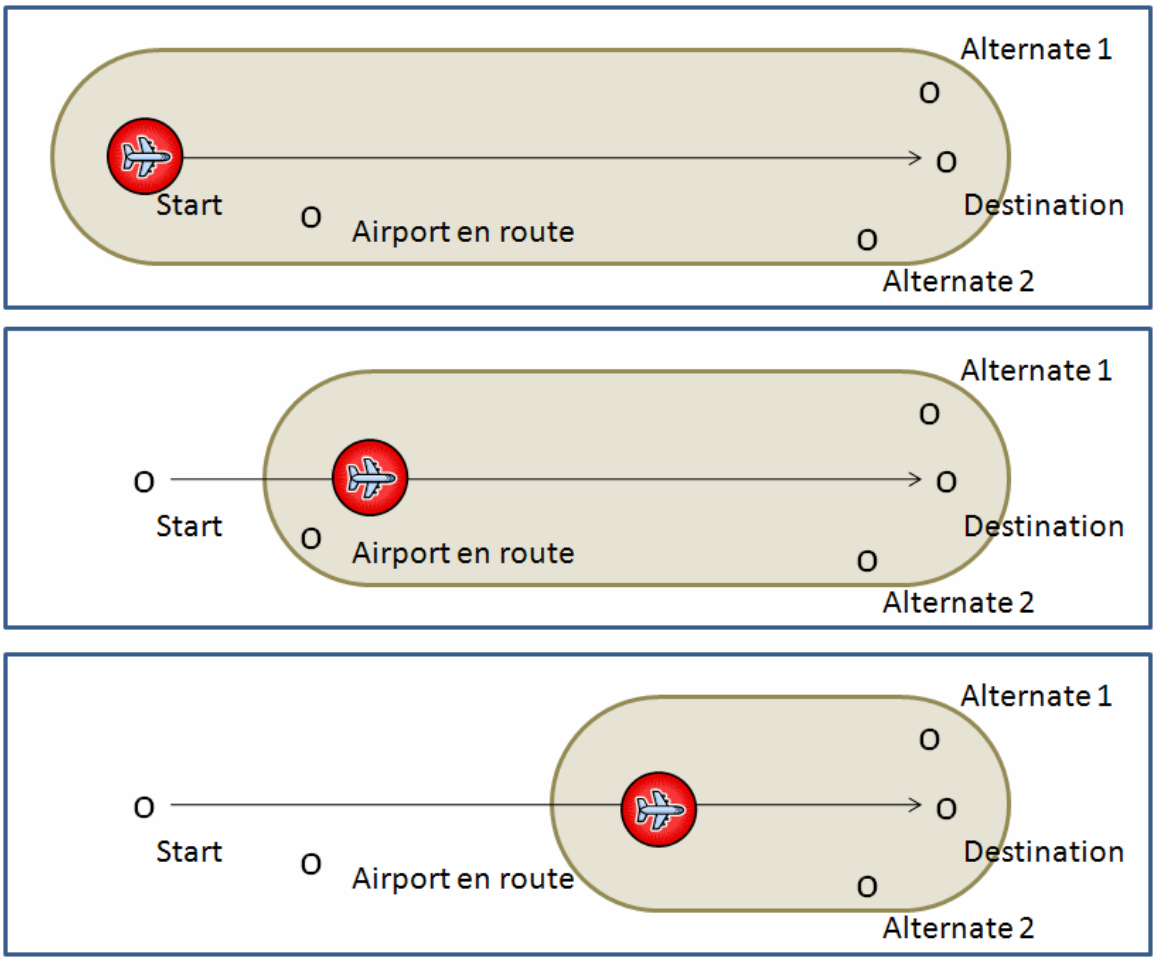

Figure 34: Spatial filtering of events with dynamically computed filter geometry

As we can see in Figure 34, the aircraft will get events from the destination airport as well as alternative airports that are used in case that some event prevents the aircraft from landing at the planned destination. In addition, events from some airports en route are received – these airports are used for emergency landings. As the aircraft progresses on its route, the filter geometry (buffer around the flight path) is updated and its size reduced. Once the aircraft has made enough progress, the updated filter ensures that events from airports that are no longer relevant (e.g. the airport en route) are not reported anymore.

Obviously, using filters with dynamically updated filter properties can lead to significant savings in both computing and transmission resources. Dynamic filter properties are supported through Event Processing. In the use cases described above, the event stream that contains the vehicle's position updates would provide the information to update the filter geometry. Timer information can govern when the geometry (the buffer in Figure 34) is updated. Event streams from other sources can then be filtered with the automatically updating geometry.

In OWS-7, the use case described above was applied in the Aviation thread. A static filter geometry was used, not a dynamically updating one. Other domains, for example the

Sensor Web domain, may also benefit from filter statements with dynamically updated properties. The integration of Event Processing functionality in OGC services should therefore be pursued in the future. Some work in this direction has already been performed – see section 9.2.3. It can serve as the basis for further developments.

9.4 Discovery of filter functionality

Various aspects are of interest when it comes to discovering which filter functionality is offered by a service. An Event Service needs to:

- indicate which filter languages it supports
- indicate which functionality of a filter language is supported
- indicate which additional capabilities are supported for a given filter language (e.g. extensions to the filter functionality – like automated unit conversion)

A service registry should harvest or be populated with the metadata of a service that contains this information. Some initial work was performed on the discovery of Event Service's filter functionality. The results are part of what is described in chapter 7. Basically, a registry was populated with a model for the OGC Filter Encoding Specification's *Filter_Capabilities* element. Sample data was used to search for event services that support certain event channels as well as certain filter functionality (see for example Listing 3).

10 Publish/subscribe specification guidelines

This chapter is about providing guidance and best practice on how to realize required parts of the Event Architecture in OGC documents and standards. The aim is to ensure that such documents consider all relevant information and document it in a common and non-redundant fashion.

The following sections provide initial guidelines on specifying events and event channels. Work in this area was started in OWS-7 – feedback from the community will help improve the specification guidelines.

Future work on publish / subscribe specification guidelines also needs to describe which information has to be provided by a specification to enable a chosen realization technology.

10.1 Specifying events

For specifying new events, the relevant information is:

- required:
    - a unique identifier for the event
    - an exact definition of the event
- optional:
    - a list of names known to be used by certain domains to label the event
    - a list of (suitable) encoding(s)

Note: different aspects of an event may be picked up by different encodings (various application schema may document the same event differently).

The following tables can be integrated in specifications to define events.

Table 39: Template for event definition table

Event Identifier [a]	Definition
identifier-value Recommendation: use URL – like *http://www.opengis.net/ows/events/X.0/EventX* Other option: combination of code space URL and name value that can easily be concatenated to a single URL – like *http://www.opengis.net/ows/events/X.0* (code space) and *EventX* (name value)	*Precise textual definition of the happening.*
a Although some values listed in the column appear to contain spaces, they shall not contain spaces.	

Table 40: Template for tabular listing of names for an event

Event Identifier [a]	Name / Label	
	codeSpace [a] (optional)	Value
event identifier-code space (optional) and value	*URI defining the codeSpace of the name / label – for example http://your-domain.net/events*	*String representing the actual value of the name or label – for example "Result Available"*
	codeSpace X	*value X*
	codeSpace Y	*value Y*
	…	*…*

a Although some values listed in the column appear to contain spaces, they shall not contain spaces.

Table 41: Template for tabular listing of suitable encodings for an event

Event Identifier [a]	Encoding		
	Namespace [a]	UML Class / XML Element Name [a]	Constraint(s) (optional)
event identifier-code space (optional) and value	*identifier of the application schema where the class / element is defined – for example "http://www.opengis.net/swes/2.0"*	*Name of the class / element that is used to encode the event – for example "SWESEvent"*	*Textual description of further encoding constraints – for example to define required values for encoded properties ("code = CAPABILITIESCHANGE" etc.)*
	namespace X	*class / element name X*	*…*
	namespace Y	*class / element name Y*	*…*
	…	*…*	*…*
event identifier Z	*namespace Z*	*class / element name Z*	*…*

a Although some values listed in the column appear to contain spaces, they shall not contain spaces.

10.2 Specifying event channels

For specifying new event channels, the relevant information is:

- required:
    - a unique identifier for the channel
    - a listing of the events that can be published on the channel and the encoding of these events
- optional:
    - an indication whether unknown events can be posted on the channel or not (default is true)

Like an event, an event channel is identified by a unique string value. We recommend using a single URL as identifier (e.g. *http://www.opengis.net/ows7/aim/channels/Weather)*. Another option is to use a combination of a code space (given as URI, e.g. *http://www.opengis.net/ows7/aim/channels*) and a string value (e.g. *Weather*).

The following table can be integrated in specifications to define which event is published on a given channel in which encoding.

Table 42: Template for defining an event channel

Event Channel Identifier [a]	**Unknown Events Published**	**Event Identifier**		**Encoding**		
		codeSpace (optional) [a]	**value** [a]	**Namespace** [a]	**UML Class / XML Element Name** [a]	**Constraint(s) (optional)**
event channel identifier--code space (optional) and value *example: http://www.opengis.net/ows7/aim/channels/Weather*	*false*	*http://www.opengis.net/ows7/aim/events*	*Forecast*	*http://www.eurocontrol.int/wx/1.1*	*Forecast*	*none*
		http://www.opengis.net/ows7/aim/events	*Observation*	*http://www.eurocontrol.int/wx/1.1*	*Observation*	*…*
		…	*…*	*…*	*…*	*…*
		…	*…*	*…*	*…*	*…*
…	*…*	*…*	*…*	*…*	*…*	*…*

a Although some values listed in the column appear to contain spaces, they shall not contain spaces.

11 Application of the Event Architecture in OWS-7

This section provides a brief overview of the OWS-7 use cases where Event Architecture aspects played a role.

11.1 Geosynchronization

The Geosynchronization service allows publication of the availability of new data or updates to existing data. The information about these events is encoded and published to interested subscribers. Different event types and channels are defined to support the controlled revision of feature updates and replication of feature stores. A possible use case is to synchronize the feature stores at different government levels (from local to national stores) – see the following figure.

Figure 35: Synchronizing feature data on various government levels – a use case for Geosynchronization

More detailed information about Geosynchronization can be found in OGC document 10-069.

11.2 Dynamic Sensor Tracking and Notification

The Sensor Fusion Enablement thread in OWS-7 investigated how the tracking of moving sensors and detection of changes in video streams can be realized using OGC standards, especially the SWE standards.

One important component of the resulting service architecture is the Tracking and Notification Service. This is a web service that offers publish / subscribe functionality. Its purpose is to continuously monitor the stream of GPS position events of a vehicle and notify the client when it detects that the vehicle entered an Area Of Interest (AOI). The change detection process offered by a WPS would then be invoked and clients notified of any detected changes – see the following figure.

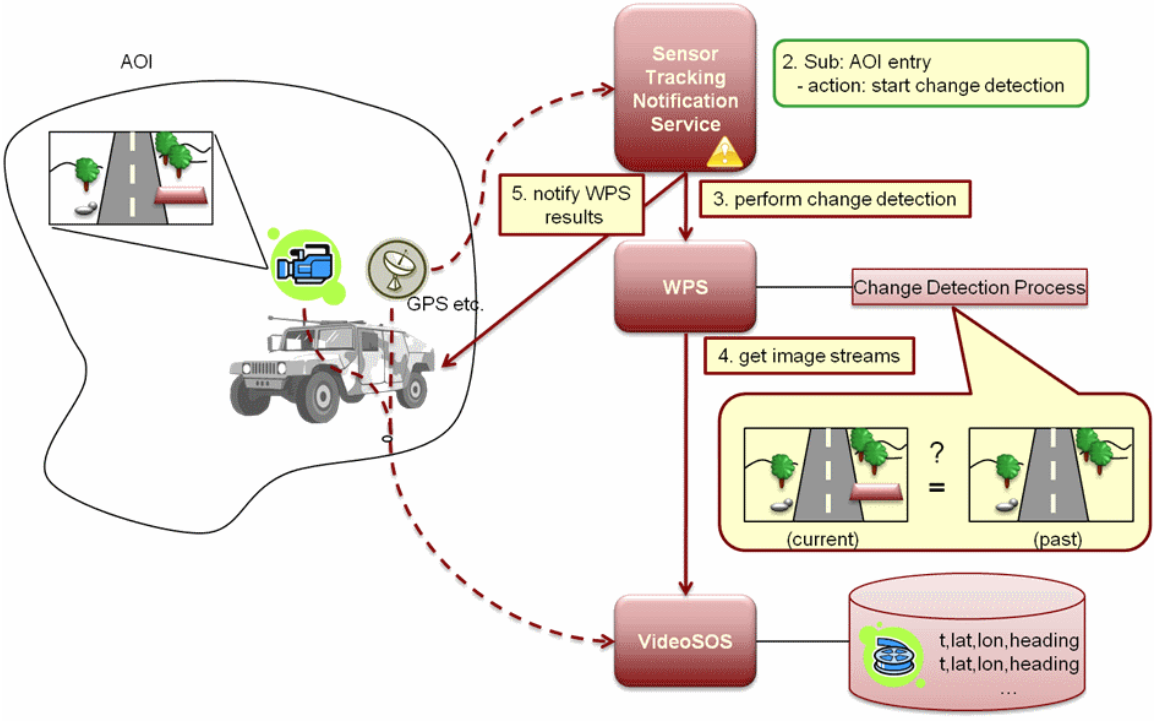

Figure 36: Dynamic sensor tracking and notification – use case of the OWS-7 SFE thread

More detailed information about dynamic sensor tracking and notification can be found in OGC document 10-061.

11.3 Aviation

The aviation thread in OWS-7 used Event Service functionality to notify subscribed clients – flight dispatchers and pilots – of events that were of interest to a planned or ongoing flight. Aeronautical events (e.g. of runway closures and special use airspace activations) as well as weather events (e.g. severe thunderstorms) that are located within a certain distance of an aircraft's flight path were detected and delivered to the respective clients. These events were then visualized and appropriate actions taken – for example to

reroute the aircraft or divert to an alternative airport. The according use case is illustrated in the following figure.

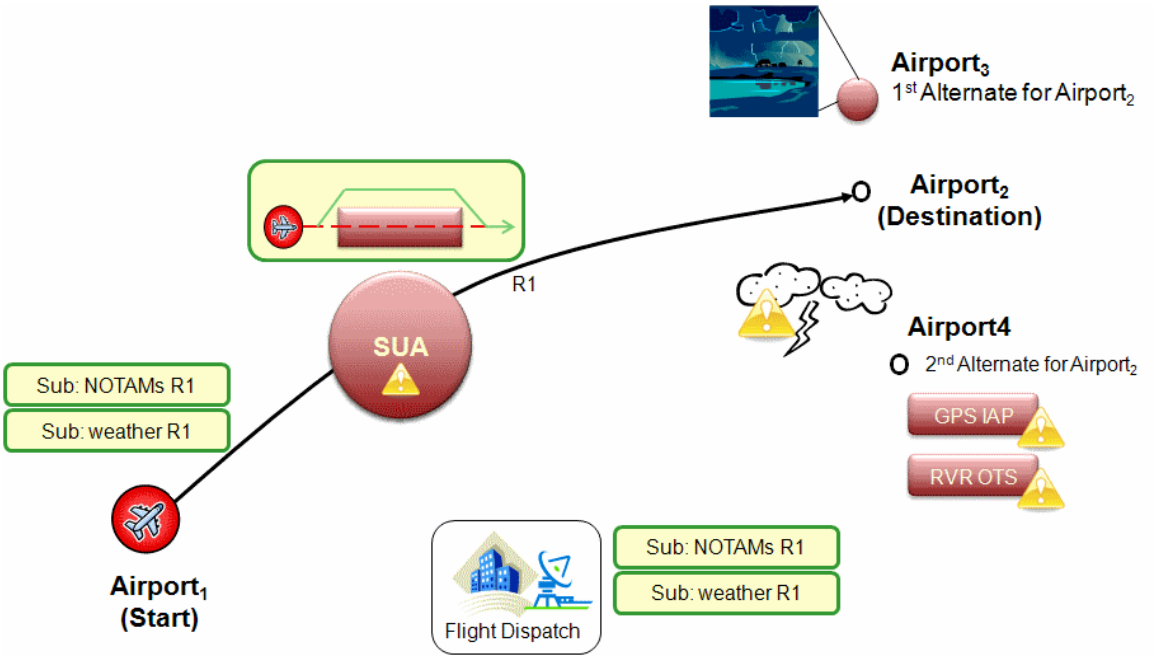

Figure 37: Monitoring flights and detecting relevant aeronautical and weather events – use case of the OWS-7 Aviation thread

More detailed information about the aviation event architecture can be found in OGC document 10-079.

12 Standards and specifications related to the Event Architecture

12.1 Service models

12.1.1 Introduction

In this section the relations between different service specifications and binding technologies that influence the event architecture or are influenced by the event architecture are introduced. The goal is to describe the connections between different specifications and to give an overview of the timeline.

12.1.2 OGC service specifications related to the Event Architecture

Here, the timeline and connection between those OGC service specifications is shown that influence the Event Architecture. These services either are designed as a kind of Event Service like the SAS or SES or they have primarily a different scope but make use of eventing technologies for specific purposes like the SPS. Figure 38 gives an overview of the timeline and the relations between the Sensor Alert Service (SAS), the Sensor Event Service (SES), the (planned) Event Service, the Event Architecture, the Web Notification Service (WNS) and the Sensor Planning Service (SPS).

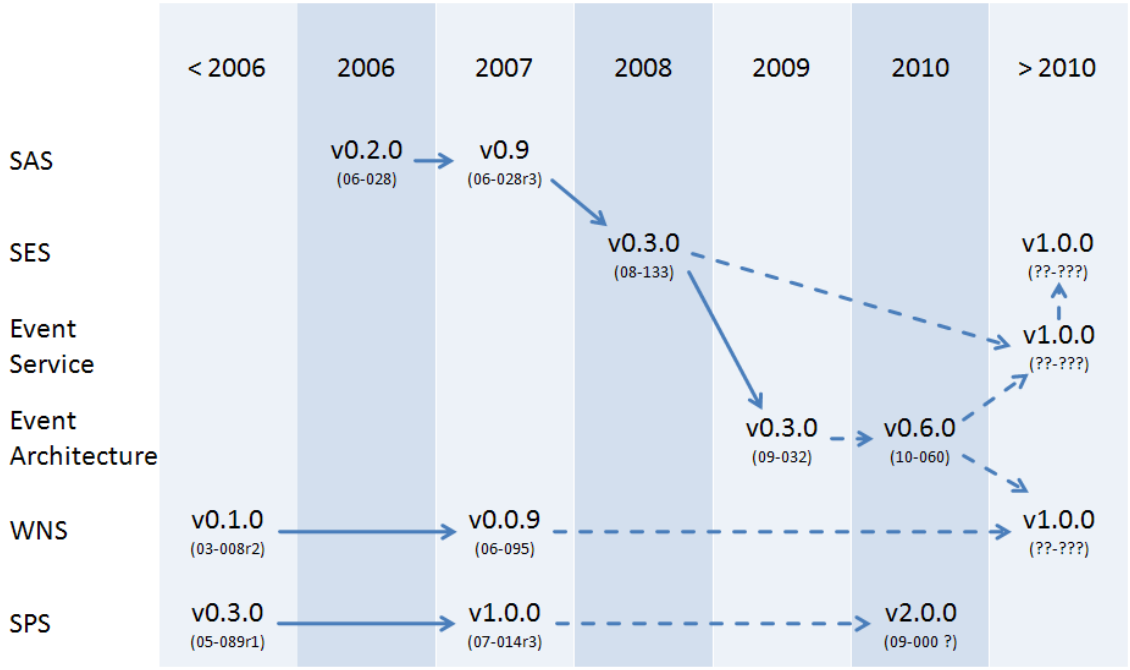

Figure 38: Overview and timeline of the relevant service specifications

The general aim of the Event Architecture work is to develop the theoretical background for an Event Service specification that is applicable in all OGC domains. This includes for instance requirement analysis, definition of terms, general architecture design but also investigation of possible realization technologies. The latter is discussed in the next section.

In the way towards an Event Service, the first steps were made in the Sensor Web Enablement (SWE) initiative with the definition of the SAS and WNS specifications. The development on these specifications went on until 2007 and resulted in two Best Practices specifications (06-028r3 for the SAS and 06-095 for the WNS). These specifications were prepared for voting but never entered the voting phase and thus, no SAS or WNS standard was released. One reason for this was the request to make stronger use of existing standards for instance for the encoding of SAS alerts but also for the service binding.

This led to the development of the SES specification, still within the SWE initiative. Like the SAS, the Sensor Event Service was designed as a broker between notification producers (e.g. sensors) and notification consumers (e.g. client applications or other services). In contrast to the SAS it reused the Web Services Notification (WS-N) standards from OASIS for the definition of the service binding. Necessary OGC and SWE operations like GetCapabilities or DescribeSensor were added on top. Also the filtering capabilities were redesigned and largely enhanced, allowing to use the OGC Filter Encoding to define the events one is interested in as well as the Event Pattern Markup Language (EML, Discussion Paper 08-132) to perform complex event processing. This service specification was released as an OGC Discussion Paper in 2008 (08-133).

In 2009 in the OWS-6 testbed the first version of the Event Architecture was developed. It took the experiences of the work on the SAS and the SES as an input but the focus was not on the development of a service specification. The resulting public engineering report (09-032) describes an abstract event architecture including the definition of important terms, an application schema for events and roles and interfaces for the abstract architecture. This abstract architecture was also mapped to multiple use cases and OGC services to show how it could be applied. In addition related problems and technologies are described like common messaging patterns, event processing, acknowledgements of events and canceling of events.

This report represents an extension and enhancement of the OWS-6 event architecture work. It defines the actual publish / subscribe functionality in much more detail. Furthermore it lays the foundation to determine which existing publish / subscribe technology should be recommended by the OGC for realizing pubsub in OWS. Based on this report and the predecessor from OWS-6 and the experiences gained from the SES a new service specification is going to be developed that fulfills the requirements developed in OWS-7 and which is applicable to all OGC domains. This service specification should therefore not be developed within the SWE initiative. Later on a new sensor specific service can be designed for instance as SWE profile of the common Event Service.

Also the WNS shall be renewed to be compatible with the Event Service. As it does not serve as a notification broker for events but rather as a protocol transducer (e.g. from HTTP to XMPP) and a service for a last mile communication it will most likely remain as a separate service specification.

As shown in Figure 38 the SPS does not have a direct influence on the Event Architecture developments. However, it makes use of notification techniques to notify users about task states. The resulting connection to the Event Architecture is described in the next section as the SPS uses the SWE Service Model (SWES) specification for notification handling.

12.1.3 Foreign standards and specifications related to the Event Architecture

In this section the relations between binding technologies and selected specifications are described. Again the timeline is presented in Figure 39. Besides the SAS, SES and the Event Architecture, the SWE Service Model (SWES) specification and the binding technologies Web Services Notification (WS-N), ATOM and Web Services Eventing (WS-E) are included.

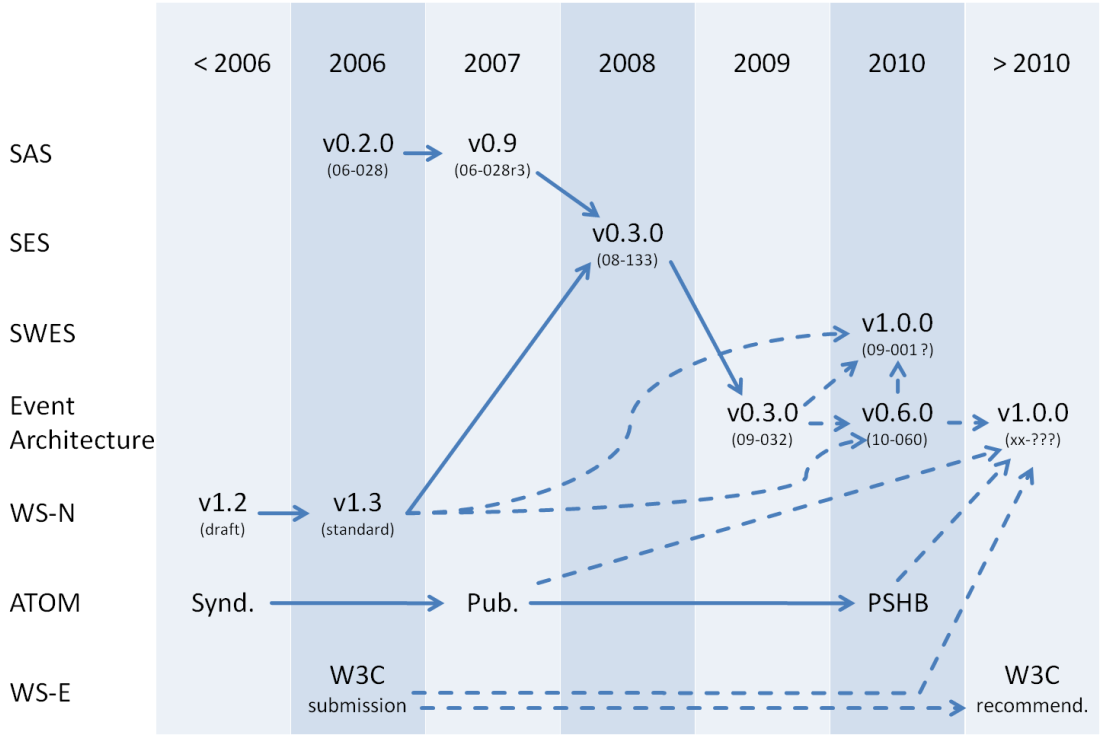

Figure 39: Overview of the timeline of selected bindings and specifications

As shown in the figure the first binding technology that influenced the OGC eventing specifications was WS-N from OASIS. It was used in the SES to define the common communication methods like Notify and Subscribe. Due to the experience made with the SES it was also selected to be used in the SWES specification for the notification handling.

The SWES specification has the goal to define standard behavior and operations across SWE service specifications similar to the OWS common specification. As said above it is used in the 2.0 versions of the SPS and the SOS. The SWES specification is currently reviewing the comments received in the RFC phase and - once all comments have been processed - will move forward to vote for adoption as an OGC Standard.

For the development of the Event Architecture also WS-E and ATOM were considered, besides WS-N. Web Services Eventing (WS-E) from the W3C has a similar scope as WS-N but differs in some aspects. In general it is a bit simpler and lighter. It is currently proposed as W3C standard but not released yet.

Under the term ATOM three specifications are listed. The first is the Atom Syndication Format (ASF) which is an XML based format for the description of lists of related information (feeds). The Atom Publishing Protocol (AtomPub) is used for editing and publishing web resources encoded as ATOM feeds. PubSubHubBub (PSHB) is a protocol that extends ATOM feeds to support push based communication via feeds instead of pulling (requesting) information updates. AtomPub and PSHB can be considered to enable a RESTful implementation of the Event Architecture and a possible OGC Event Service.

Both the latest WS-Eventing and Atom specifications need to be investigated in detail and the results incorporated in a further revision of the OGC Event Architecture.

12.2 Information models

12.2.1 Introduction

In this section various information models are described that can be used for the encoding of events.

12.2.2 Timeline

The following figure (Figure 40) shows the relations and the timeline of multiple information models. These include information models developed by the OGC like the Geography Markup Language (GML), Observations and Measurements (O&M) and SWE Common but also external information models like the Aeronautical Information Exchange Model (AIXM), the Weather Information Exchange Model (WXXM) and the Common Alerting Protocol (CAP). Please note that not all relations between GML and other models are included in the figure for readability reasons. The relations shall be interpreted to be transitive, for instance AIXM 5.1 is also using GML 3.2.1 because it is based on AIXM 5.0 which uses GML 3.2.1 directly.

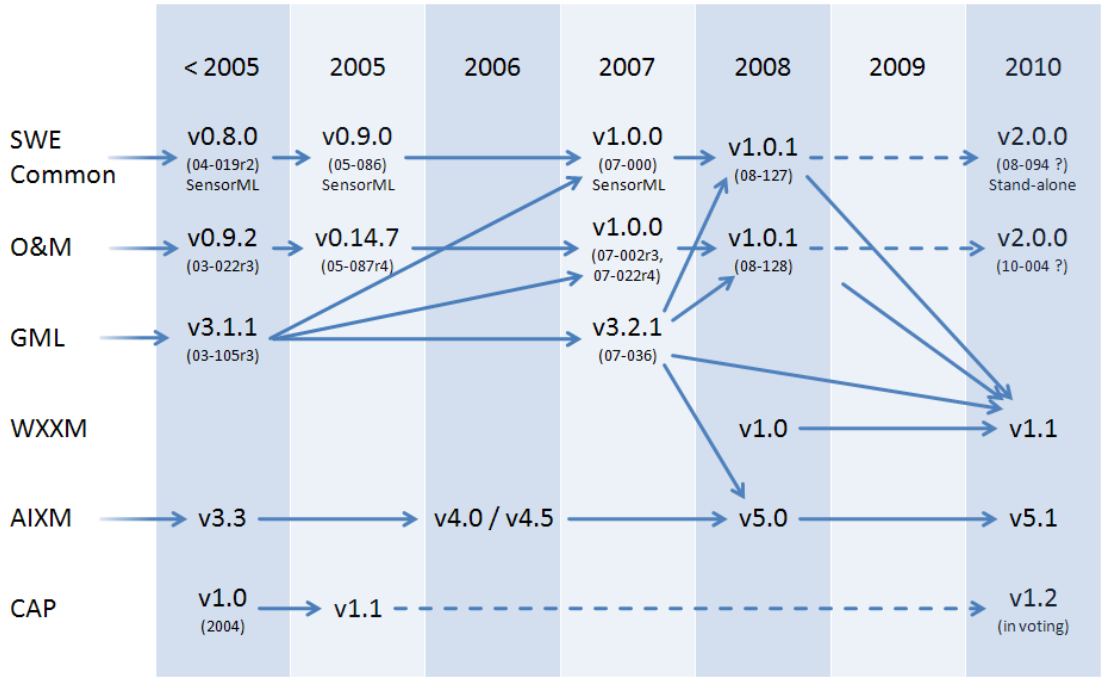

Figure 40: Overview of the timeline and relations of selected information models

12.2.3 GML

The Geography Markup Language is widely used for the encoding of geographical features. It defines the encoding of a wide range of properties like the geometry, reference system information, temporal properties and other data types. GML therefore provides the baseline for other information models like O&M. In fact, O&M is a GML application schema, meaning that it is encoded following encoding rules defined by GML. The standard output of Web Feature Services is usually encoded in GML.

GML does not define data types itself for encoding events. A given application schema has to define the encoding of its events. This can but does not have to leverage the Event Model that was first described in OWS-6 Event Architecture ER (09-032). Note that the encoding of an event shall – according to the Event Architecture – provide the information to determine the time when the event happened, that is the event time.

12.2.4 Event Model (Revision)

Based upon the OGC Event Definition developed in OWS-6, the first version of a GML Application Schema for events - the Event Model - was developed. This model defines the structure of basic event types. In addition, some special events with common purpose are provided. The Event Model was first documented in the *OWS-6 SWE Event Architecture Engineering Report* (OGC 09-032).

In OWS-7, the following improvements to the model were identified:

- Each event shall have an optional spatialExtent property. This property shall be populated with the geometry that best describes the spatial boundary of the event. Of course, the property can only have a value if the event has a spatial context and if that context can be expressed as a geometry.

- An event shall support the addition of a number of soft-typed properties. These named value attributes support the inclusion of any property that is defined through an extension, added ad hoc or used for testing purposes. The name of such an attribute shall be well-defined, ideally via a unique URL, so that clients can unambiguously identify the purpose of a given soft-typed property.

- The target property of the EventEventRelationship shall not be constrained to be a GML AbstractFeature. It shall be possible to point to any existing type. This for example supports use cases in which the members of a DerivedEvent are CAP alerts.
  This change required that the EventEventRelationship type does not derive from the EventFeatureRelationship type. Both types could derive from a common relationship type that defines the role property. This change has not been required yet.

Figure 41 shows the revised model that incorporates these changes.

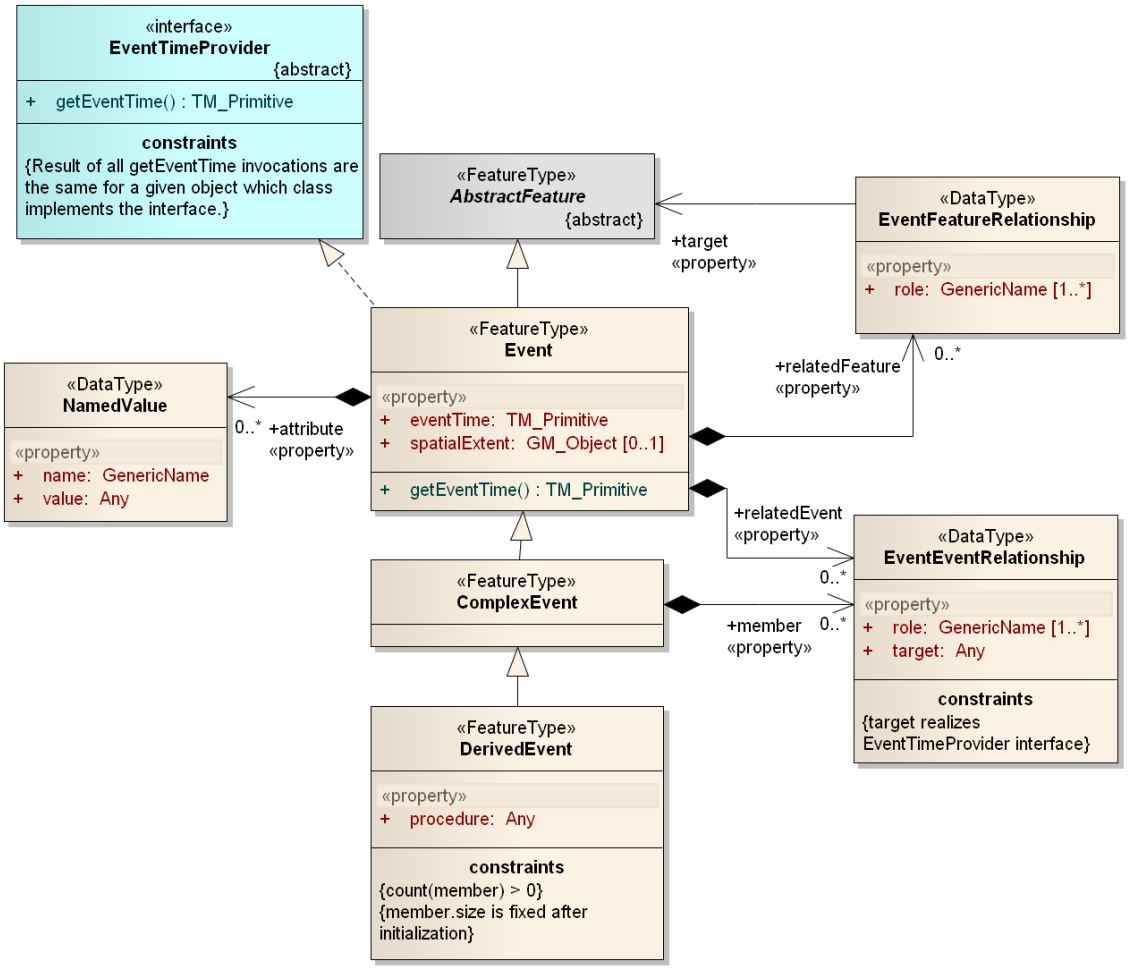

Figure 41: Revision of the Event Model package that defines core event types

The XML Schema implementation of the revised Event Model is provided in *Annex E – XML Schema for Event Model Revision*.

12.2.5 O&M

O&M is a GML application schema. It is currently available as an OGC standard in version 1.0 which is based on GML 3.1.1, a version based on GML 3.2.1 (1.0.1) is available as discussion paper and can be used in order to be compatible to other information models using GML 3.2.1. As for many other SWE standards version 2.0 is currently under development.

O&M is used to encode an observation or measurement as a feature. Besides the measured result also the observed feature (e.g. a lake), the observed property (e.g. the pH value), the observation procedure (e.g. a pH sensor description) and the sampling time must be available. Because the result time in an O&M measurement provides the time when the observation actually produced the result, that is *when the result generation happened*, an O&M observation is an event according to the definition of the Event Architecture (the result time being the event's event time).

Also spatial filtering via the observed feature (feature of interest) or on metadata properties (observed property and procedure) can be performed. Filtering on the measured value itself may become problematic as the encoding for the measurement result is not defined in the O&M specification.

12.2.6 SWE Common

SWE Common defines common data types for the use in SWE. These types are derived from the GML Abstract Type but do not reuse the types defined in GML like gml:Quantity. The SWE Common data types are newly defined instead having some important differences making them incompatible to the GML types. One instance of this difference is the optionality of the actual value. This allows using the SWE Common types as a template for values instead of having to include a specific value itself. More information on the differences between SWE Common and GML can be found in the OWS-6 SWE Information Model Harmonization ER (09-031r1).

Similar to O&M, SWE Common relies on GML 3.1.1 in its 1.0 version. A Discussion Paper for a GML 3.2.1 support is available, too and the 2.0 version will support GML 3.2.1 directly. Another important difference between version 2.0 and the previous versions is that SWE Common will be extracted from the SensorML specification and released as an independent standard.

SWE Common alone is not well suited as an event encoding as it just defines data types. However it can be embedded in other encoding like O&M and serve as encoding for the measurement result. It is also possible to define a complex SWE Common data type including mandatory temporal properties and use this as event encoding.

12.2.7 AIXM

The Aeronautical Information Exchange Model is an XML based encoding for aeronautical features (like runways, navaids and many more). It is developed by the Federal Aviation Administration (FAA) and Eurocontrol. Since version 5.0 it is based on GML 3.2.1 and represents the aeronautical features as extensions of GML features. With respect to the continuous changes to feature properties (e.g. runway closure, …) the AIXM features are extended with a temporal validity model based on time slices. Besides the basic information it is possible to encode temporal and permanent changes to the data and keep track of these changes.

The support of event encoding is one important aspect in AIXM as one function of AIXM shall be the enablement of digital NOTAMs (Notice To AirMen). A NOTAM is a notification for pilots to inform them about changed conditions for instance at an airport. AIXM includes specific elements for the encoding of the NOTAMs which also can serve as encoding for (aeronautical) events. More information on the use of AIXM and eventing aspects can be found in the OWS-6 and -7 Aviation Engineering Reports (09-050r1 and 10-079).

12.2.8 WXXM

The Weather Information Exchange Model is also based on XML. It is used to encode weather observations (current observations and forecasts). These are represented as an extension of O&M observations. In version 1.1 it uses O&M version 1.0.1 to be compatible to the AIXM which relies on GML 3.2.1. It is usable in the same way as O&M for the encoding of events but adds additional properties that help to filter on properties that are of special interest in the aviation domain.

12.2.9 Common Alert Protocol

12.2.9.1 Introduction

The Common Alert Protocol (CAP) is a message format for encoding alert or notification messages.

The main idea of CAP is to normalize the expression of alert messages arising from various sources. Such normalization then allows these messages to be aggregated and compared to aid situational awareness and pattern detection.

To that end, the CAP standard describes a document object model that defines the structure of an alert or notification messages and a protocol that describes how CAP messages should be constructed, used and related to each other. The CAP standard also describes a concrete implementation of its document model using XML and expressed in XML-Schema.

A limited amount of testing was done by CubeWerx Inc during the OWS-7 test bed with CAP v1.1 and the Geosynchronization service. That work involved generating CAP messages as notification messages sent to subscribers. So rather than sending subscribers ATOM entries, the format used in the GSS specification, CAP messages were generated and sent instead. Unfortunately, it was not possible in the time frame of the project to also test filtering of CAP message using the Filter encoding specification (as was done with ATOM entries) so any discussion about using FES with CAP is purely notional.

12.2.9.2 Document object model

Figure 42 illustrates the UML definition of a CAP message.

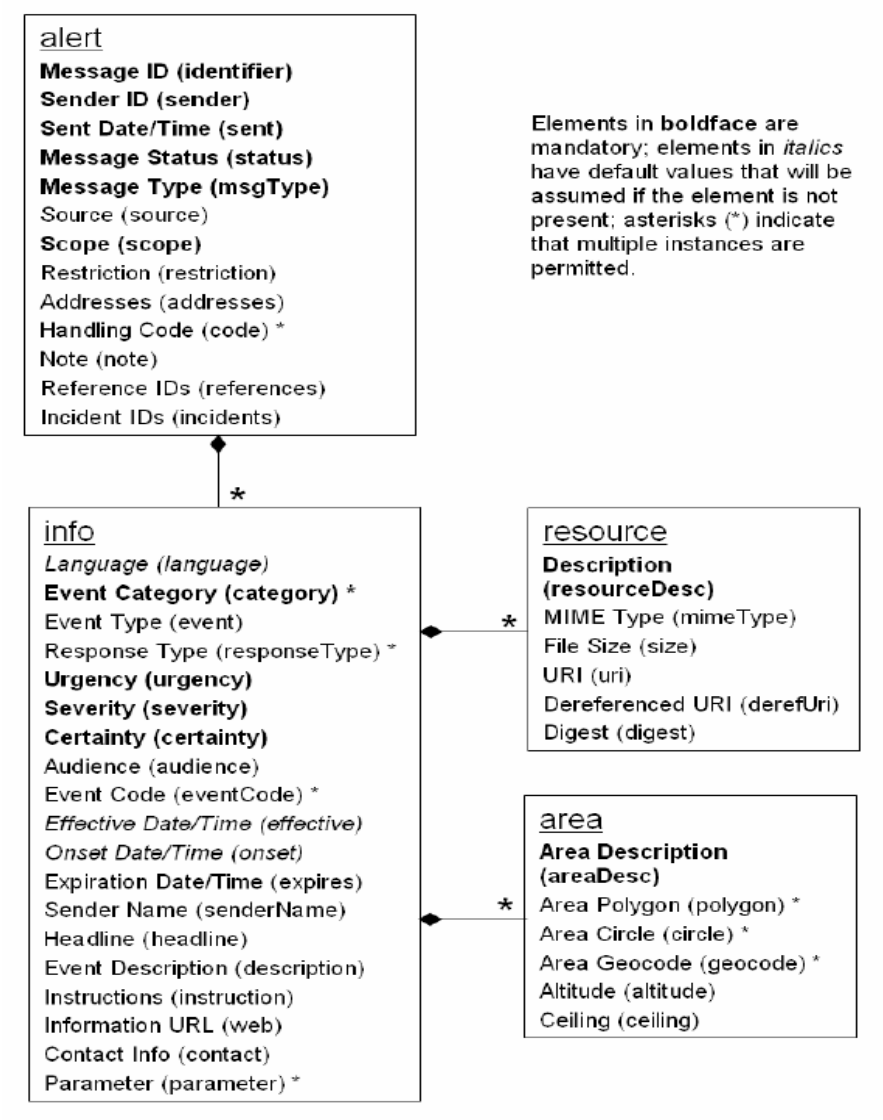

Figure 42: UML for CAP Message

The model defines four primary "segments": **alert**, **info**, **resource** and **area**. The alert segment provides basic information about the message: purpose, source, status, etc. The alter segment can be used alone for message acknowledgements, cancellations and other system functions. The info segment encodes the details of the alert. Multiple info segments can be included to express information in multiple languages or across multiple disconnected areas. The resource segment provides an optional reference to additional related resources. The area segment describes the geographic area to which the notification applies.

12.2.9.3 Spatially enabled

CAP is spatially enabled allowing messages to be targeted geographically.

The following XML-Schema fragment shows the declaration of the **cap:area** element which allows CAP messages to be tagged with location.

Listing 5: Extract of CAP XML Schema showing location support

```
<element name = "area" minOccurs = "0" maxOccurs = "unbounded">
<complexType>
   <sequence>
      <element name = "areaDesc" type = "string"/>
      <element name = "polygon" type = "string" minOccurs = "0" maxOccurs = "unbounded"/>
      <element name = "circle" type = "string" minOccurs = "0" maxOccurs = "unbounded"/>
      <element name = "geocode" minOccurs = "0" maxOccurs = "unbounded">
         <complexType>
            <sequence>
               <element ref = "cap:valueName"/>
               <element ref = "cap:value"/>
            </sequence>
         </complexType>
      </element>
      <element name = "altitude" type = "string" minOccurs = "0"/>
      <element name = "ceiling" type = "string" minOccurs = "0"/>
   </sequence>
</complexType>
```

Location is expressed textually using the **cap:areaDesc** element and can be expressed geometrically or using a geo-code such as a postal code. The standard states that geometric tagging is the preferred method. Unfortunately, the geometric tagging is done using a non-standard encoding forcing a conversion to and from a standard encoding like GML.

12.2.9.4 Code lists

The CAP standard defines a number of lists of values for parameters in the model. Here are a couple of examples:

```
<element name = "category" maxOccurs = "unbounded">
   <simpleType>
      <restriction base = "string">
         <enumeration value = "Geo"/>
         <enumeration value = "Met"/>
         <enumeration value = "Safety"/>
         <enumeration value = "Security"/>
         <enumeration value = "Rescue"/>
         <enumeration value = "Fire"/>
         <enumeration value = "Health"/>
         <enumeration value = "Env"/>
         <enumeration value = "Transport"/>
         <enumeration value = "Infra"/>
         <enumeration value = "CBRNE"/>
         <enumeration value = "Other"/>
      </restriction>
   </simpleType>
</element>

<element name="responseType" minOccurs="0" maxOccurs="unbounded">
   <simpleType>
      <restriction base="string">
         <enumeration value="Shelter"/>
         <enumeration value="Evacuate"/>
         <enumeration value="Prepare"/>
         <enumeration value="Execute"/>
         <enumeration value="Monitor"/>
         <enumeration value="None"/>
      </restriction>
   </simpleType>
</element>
```

These lists of values are hard-coded into the schema significantly reducing the extensibility of the model. Also, the particular vocabulary used is focused on emergency management which is fine if that is the intended context within which the notification messages are being used. Otherwise, the enumeration value **Other** tends to be used a lot significantly diminishing the usefulness of a parameter like **category**. This was certainly the case when CAP was used in the GSS experiment.

A better approach would have been to use code lists similar to the approach used in GML with the **gml:CodeType** type. This allows lists of values to be dynamically referenced allowing the vocabulary to be adjusted to fit the particular context. Also, from the point of view of implementing a generic CAP client, code lists allow such clients to read these lists and generate lists at run-time thus adjusting the client to the context.

12.2.9.5 Resource references

The CAP message format defines the **cap:resource** element which may be used to embed or reference resources related to the notification. This was intended to allow references to images, audio and/or video content but was used effectively during the GSS experiment to reference, for example, modified features. The element could also contain a more detailed representation of the event that the CAP alert summarizes information of.

12.2.9.6 Other features

The following list describes features of CAP that are anticipated to be useful in a general eventing environment but which were not exercised by the GSS during the OWS7 test bed:

- Multilingual support - alert messages can include the same information in multiple languages
- Multi-audience support - alert messages can be targeted to multiple audiences
- Message management - the protocol defines procedures for updating and canceling previously issued alerts
- Time - alert messages include onset and expiry times which allow for phased of delayed alerts to be signaled

12.2.9.7 Interoperability

Interoperability with respect to different ways of encoding notifications can be achieved by using a gateway that provides uniform access to various notification technologies - the Web Notification Service (OGC best practice) could become such a gateway (right now it is limited, unfortunately). The gateway can then tie into emergency alert middleware that uses CAP to route alert messages to recipients.

12.2.9.8 Geosynchronization

One of the services in the OWS-7 test bed that implemented notification was the Geosynchronization service. The GSS uses ATOM as the preferred event encoding format but the specification does not specifically mandate which format shall be used to encode notification messages sent to subscribers.

During the OWS 7 test bed, both ATOM and CAP were used for encoding notification messages sent to subscribers. The choice of which format a subscriber would receive was made using an "outputFormat" parameter on the subscribe request.

The following listing illustrates a notification message, encoded using CAP, generated by a GSS and sent to a subscriber when a new event (i.e. entry) appears in one of the service's event channels (i.e. feeds):

Listing 6: CAP encoded notification generated by ESS

```
<cap:alert
   xmlns:cap="urn:oasis:names:tc:emergency:cap:1.1">
   <cap:identifier>urn:uuid:885e061e-b6dd-4a4b-ad15-8131e61c512a</cap:identifier>
   <cap:sender>geosync@cubewerx.com</cap:sender>
   <cap:sent>2010-06-03T20:56:41-05:00</cap:sent>
   <cap:status>Actual</cap:status>
   <cap:msgType>Alert</cap:msgType>

<cap:source>http://www.cubewerx.com/ows7/cubeserv/cubeserv.cgi?service=GSS&amp;request=Ge
tCapabilities</cap:source>
   <cap:scope>Public</cap:scope>
   <cap:info>
      <cap:category>Other</cap:category>
      <cap:event>NEW REPLICATIONFEED ENTRY</cap:event>
      <cap:responseType>None</cap:responseType>
      <cap:urgency>Future</cap:urgency>
      <cap:severity>Unknown</cap:severity>
      <cap:certainty>Observed</cap:certainty>
      <cap:eventCode>
           <cap:valueName>action</cap:valueName>
           <cap:value>INSERT</cap:value>
      </cap:eventCode>
      <cap:effective>2010-06-03T20:56:41-05:00<cap:effective>
      <cap:senderName>CuebWerx Inc.</cap:senderName>
      <cap:headline>New Feature Available</cap:headline>
      <cap:description>A new feature with identifier CWFID.BUILDINGGEOSURFACE.0.326 has
been processed by the GSS and is now available.</cap:description>
      <cap:instruction>The new feature can be retrieved by following the uri for the
"Feature Type" resource.  The WFS transaction that created the feature can be obtained by
following the uri for the "ATOM entry" resource and reading the content of the
entry.</cap:instruction>
      <cap:resource>
         <cap:resourceDesc>ATOM entry</cap:resourceDesc>
         <cap:mimeType>application/atom+xml</cap:mimeType>
<cap:uri>http://www.cubewerx.com/ows7/cubeserv/cubeserv.cgi?service=GSS&amp;request=GetEn
tried&amp;id=urn:uuid:8084a1cd-50c6-49b1-9641-96241ad5dd1e</cap:uri>
      </cap:resource>
      <cap:resource>
         <cap:resourceDesc>Feature Type</cap:resourceDesc>
         <cap:mimeType>application/gml+xml; version-3.1.1</cap:mimeType>
<cap:uri>http://www.pvretano.com/nga/cubeserv.cgi?service=WFS&amp;CONFIG=plugweek&amp;DAT
ASTORE=tds&amp;request=GetFeature&amp;typeName=BUILDINGGEOSURFACE&amp;featureId=CWFID.BUI
LDINGGEOSURFACE.0.326</cap:uri>
      </cap:resource>
      <cap:area>
         <cap:polygon>18.5778097 -72.2787128 18.5778097 -72.2787692 18.5778911 -
72.2787692 18.5778936 -72.2788604 18.5777792 -72.2788631 18.5777741 -72.2787128
18.5778097 -72.2787128</cap:polygon>
      </cap:area>
   </cap:info>
</cap:alert>
```

12.2.9.9 Filtering CAP messages

As illustrated in Listing 6 and section 12.2.9.8, a CAP message is composed of values that are encoded in-line (in the alert, info and area segments) and of values that may be remotely references (in the resource segment).

Filtering CAP messages using FES to constrain in-line values is straight forward and does not present any technical challenges. In fact, CAP and FES work quite well together especially because CAP messages are spatio-temporally enabled thus allowing the spatial and temporal operators defined in FES to be used in query predicates when geometric location tagging is used. Of course, there is the issue of converting the CAP geometry to GML (see 12.2.9.3) but this is more a nuisance than a show stopper.

Filtering on referenced values presents more of a challenge. Filtering on referenced values is not a function of FES but of the service within which FES is embedded – see section 9.1.3. If the service is capable of resolving referenced values then filtering is possible.

Perhaps the best analogy to use here is the Web Feature Service 2.0 implementation specification. The WFS 2.0 standard defines a set of request parameters called the "Standard Resolve Parameters". The following XML Schema fragment from the WFS 2.0 schemas defines these parameters:

```
<xsd:attributeGroup name="StandardResolveParameters">
   <xsd:attribute name="resolve" type="wfs:ResolveValueType" default="none"/>
   <xsd:attribute name="resolveDepth" type="wfs:positiveIntegerWithStar"
      default="*"/>
   <xsd:attribute name="resolveTimeout" type="xsd:positiveInteger"
      default="300"/>
</xsd:attributeGroup>
<xsd:simpleType name="ResolveValueType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="local"/>
      <xsd:enumeration value="remote"/>
      <xsd:enumeration value="all"/>
      <xsd:enumeration value="none"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="positiveIntegerWithStar">
   <xsd:union memberTypes="xsd:positiveInteger wfs:StarStringType"/>
</xsd:simpleType>
<xsd:simpleType name="StarStringType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="*"/>
   </xsd:restriction>
</xsd:simpleType>
```

The resolve parameters appear on requests and direct the service how to handle referenced values. The "resolve" parameter tells the service which referenced values to resolve; local, remote, all or none. The "resolveDepth" and "resolveTimeout" parameters control the number of nested levels to which referenced values should be resolved and how long to wait to resolve a remote value reference. When processing a predicate and the service encounters a remote value reference, it uses these parameters to control how it resolves that reference in order to obtain a value to test in the predicate. If the resolution is successful and a testable value is obtained, filter processing continues as normal. If value resolution fails, then filter processing fails and an exception is raised.

12.2.9.10 EDXL-DE

EDXL-DE (Emergency Data Exchange Language Distribution Element) was not used in the GSS experiment and so it is discussed only briefly here.

The purpose of EDXL-DE is to facilitate the routing of any XML emergency message to recipients. An EDXL-DE message may be thought of as a "container". An EDXL-DE message provides the information to route "payload" message sets (such as Alerts or Resource Messages), by including key routing information such as distribution type, geography, incident, and sender/recipient IDs.

The relationship between EDXL-DE and CAP is similar to the relationship between SOAP and WFS. Like a WFS message that is embedded within the body of a SOAP message, a CAP message can be the payload within an EDXL-DE message. Because CAP messages already encode routing or targeting information (e.g. cap:addresses, cap:area, etc.) there is an non-trivial amount of overlap in the kind of information that an EDXL-DE message encodes when compared to the kind of information encoded in a CAP message.

Clients considering using CAP as event encoding should therefore clearly identify which information an event needs to contain. If the CAP model provides the properties to encode the required information without at the same time requiring too much additional information (that the client does not need), CAP can be used as the event encoding. Otherwise, the development of a more focused encoding is a viable option. Such events can still be inserted into an EDXL-DE container object to leverage routing functionality offered by emergency messaging middleware (see section 12.2.9.7).

13 Annex A - Publish Subscribe Requirements

This Annex contains the documentation of the requirements that were derived from the abstract publish / subscribe model (see section 6.2). The requirements are grouped into packages as described in section 6.4.

The following subsections first give a graphical overview of the requirements of each package, structured into modules. The requirements with their normative statements are then listed in a table.

13.1.1 Resource Requirements Package

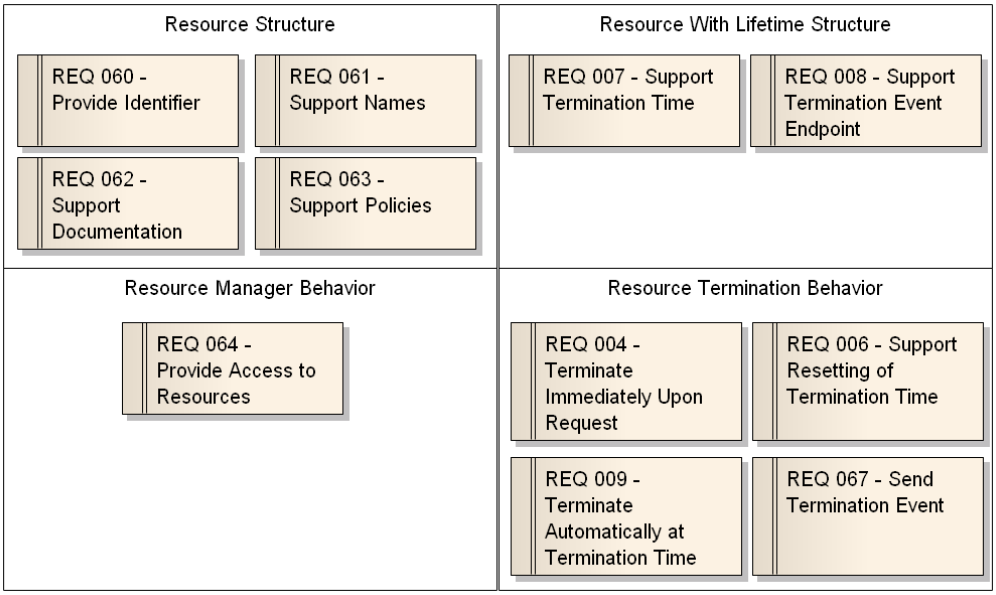

Figure 43: Resource Requirements

Table 43: Resource Requirements Details

Requirement Number	Requirement Normative Statement
004	A resource with lifetime shall provide an operation to explicitly and immediately terminate the resource.
006	A resource with lifetime shall support an operation to renew its termination time to a given time instant.
007	A resource with lifetime shall support the provision of the point in time when the resource will automatically be terminated.

Requirement Number	Requirement Normative Statement
008	A resource with lifetime shall support the provision of an endpoint to which a termination event shall be sent when the resource is terminated.
009	A resource with lifetime shall support the automatic termination of itself once the termination time set for the resource is in the past.
060	A resource shall support an (explicit or implicit) identifier. This can for example be a URL or WS-Addressing endpoint.
061	A resource shall support multiple resource names that can be assigned to it.
062	A resource shall support the inclusion of documentation about the resource.
063	A resource shall support the inclusion of policies that define behavior that is required or requested.
064	A system entity acting as a resource manager shall provide an operation to retrieve one of its resources via its identifier.
067	Before termination of a resource with lifetime, the resource or the entity managing the resource shall publish a termination event to the endpoint provided in the resource.

13.1.2 Consumer Requirements Package

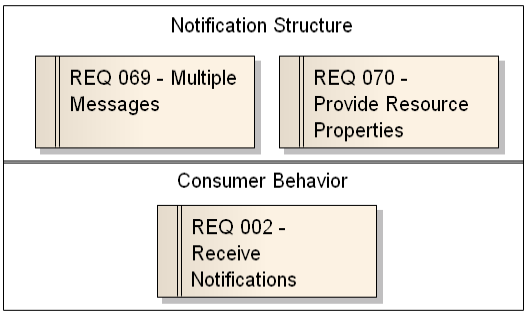

Figure 44: Consumer Requirements

Table 44: Consumer Requirements Details

Requirement Number	Requirement Normative Statement
002	A system entity acting as a consumer shall provide an operation to receive a notification.
069	A notification shall support the inclusion of one or more messages.
070	A notification shall provide the properties of a resource (an implicit or explicit identifier, names etc).

13.1.3 Publish Subscribe Requirements Package

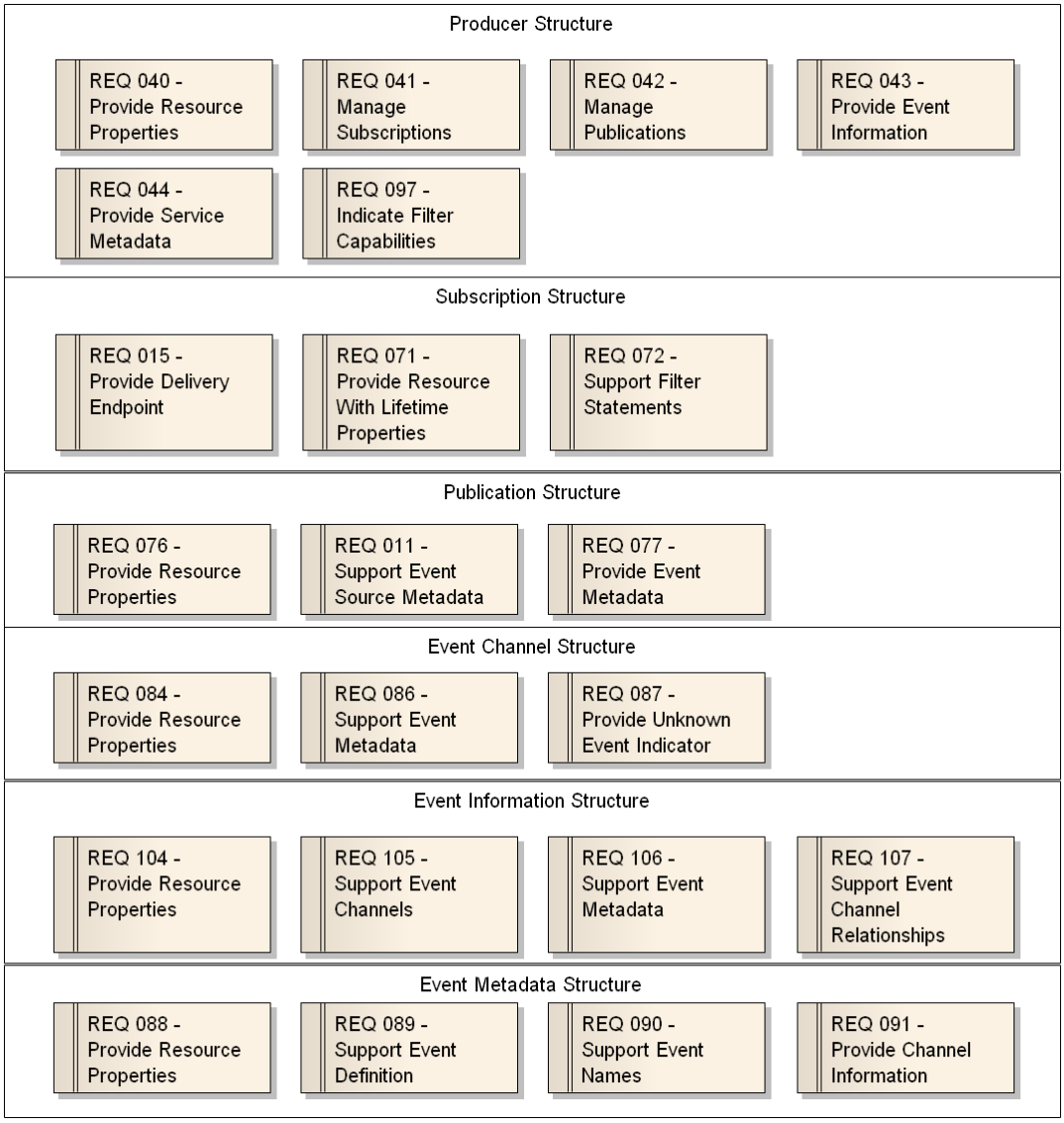

Figure 45: Publish Subscribe Structural Requirements

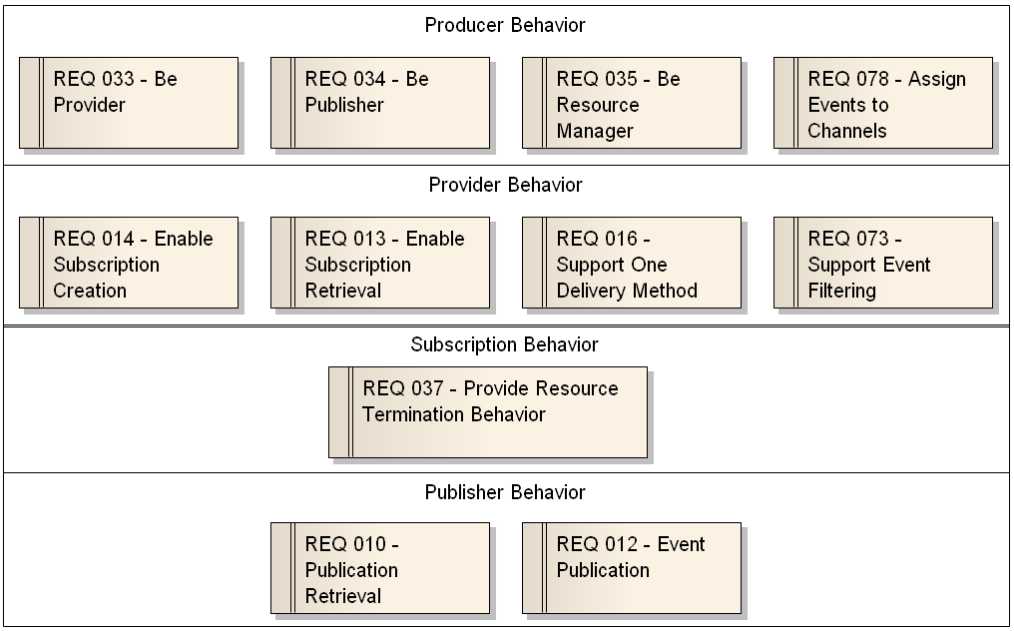

Figure 46: Publish Subscribe Behavior Requirements

Table 45: Publish Subscribe Requirements Details

Requirement Number	Requirement Normative Statement
010	A system entity acting as a publisher shall provide an operation to get all existing publications.
011	A publication shall support the provision of metadata about the entity that generates the published events.
012	A system entity acting as a publisher shall deliver events to interested consumers using the operation defined for a consumer to receive notifications. Note: interest may but does not necessarily have to be expressed through a subscription.
013	A provider shall provide an operation to get all existing subscriptions.
014	A provider shall offer an operation to express interest in delivery of certain events. It shall therefore enable the creation of subscriptions.
015	A subscription shall provide information on an endpoint where events matching the subscription's filter criteria will be sent to.

Requirement Number	Requirement Normative Statement
016	A provider shall support push and / or pull based notification delivery to a given endpoint. A push endpoint shall be explicitly provided by a subscriber while a pull endpoint shall be created and assigned to a subscription by the provider.
033	A producer shall be a provider, that is enable creation of subscriptions and provide access to them.
034	A producer shall be a publisher, that is publish events matching subscription criteria to the delivery endpoint(s) of these subscriptions and provide access to the according publications. A producer shall start matching and sending events to a consumer once the subscribe operation for this consumer was finished successfully.
035	A producer shall be a resource manager, that is provide access to the resources - publications and subscriptions - it manages.
037	A subscription shall provide the termination behavior defined for a resource with lifetime.
040	A producer shall support the properties that a resource offers (like being identifiable and to support policies).
041	A producer shall provide information on the subscriptions it is currently managing.
042	A producer shall provide information on the publications it is currently managing.
043	A producer shall provide information on the events it publishes and the channels it exposes.
044	A producer shall provide metadata about its capabilities.
071	A subscription shall support the properties that a resource with lifetime offers. This includes resource properties like a resource identifier and indication of termination time.
072	A subscription shall support the provision of filters which are used to express the interest in a certain set of published events.
073	A provider shall determine the interest of a subscription into a set of events based upon the filter statements of the subscription. If no filter statement is provided then interest into all events shall be

Requirement Number	Requirement Normative Statement
	assumed. Note: a filter may in some realization technologies be implicit, e.g. when consumers are subscribed for events published in web feeds; there, the filter to receive events only from the specific channel / feed is implied in the subscription.
076	A publication shall provide the properties of a resource (an implicit or explicit identifier, names etc).
077	A publication shall provide metadata about the events that are generated as part of the publication.
078	A producer shall ensure that events are published on its event channels according to the channel definitions.
084	An event channel shall support the properties that a resource offers (like being identifiable and supporting names).
086	An event channel shall support the provision of information on the events published on that channel. Furthermore, it shall support the provision of the encoding(s) in which these events are published on the channel.
087	An event channel shall provide a boolean property that indicates whether events not listed in the channel event metadata can be published on the channel or not.
088	Event metadata shall support the properties that a resource offers (like being identifiable and supporting names).
089	Event metadata shall support the provision of a detailed event definition.
090	Event metadata shall support the provision of names that domains have assigned to the represented event.
091	Event metadata shall provide information on which channel(s) the event is published. Furthermore, it shall support the provision of information on the encoding(s) in which the event is published on a given channel.
097	A producer shall indicate which filter functionality it offers to subscribers.
104	Event information shall support the properties that a resource offers (like supporting documentation).

Requirement Number	Requirement Normative Statement
105	Event information shall support the provision of information on the event channels that are exposed by a service.
106	Event information shall support the provision of metadata on the events that are published by a service.
107	Event information shall provide data on which events are published on which event channels in which encoding(s).

13.1.4 Registrar Requirements Package

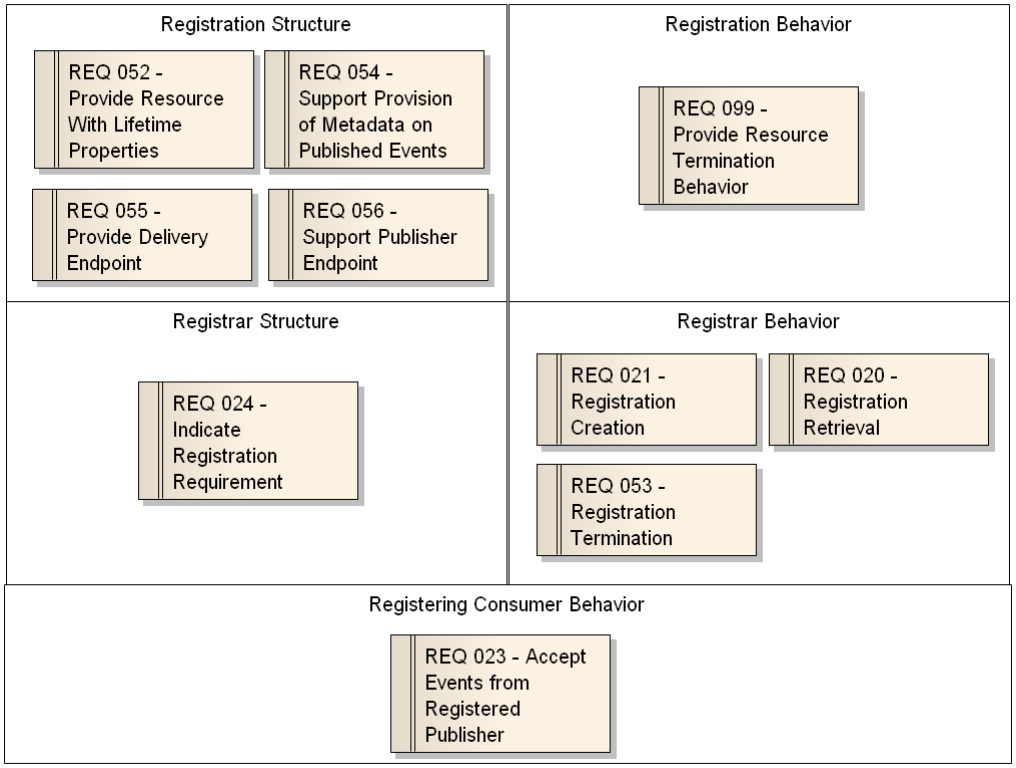

Figure 47: Registrar Requirements

Table 46: Registrar Requirements Details

Requirement Number	Requirement Normative Statement
020	A registrar shall provide an operation to get all registrations it

Requirement Number	Requirement Normative Statement
	currently manages.
021	A registrar shall offer an operation to register a new publisher at a consumer.
023	A system entity that implements registrar functionality to restrict access to its consumer functionality shall accept each notification from a registered publisher unless the notification does not match the registration information (e.g. if events other than those described in the registration are sent). Note: a consumer can choose to accept notifications from publishers that were not registered.
024	A system entity that requires publishers to register before they can send events to it shall indicate this behavior via a policy statement.
052	A registration shall support the properties that a resource with lifetime offers. This includes resource properties like a resource identifier and indication of termination time.
053	A registrar shall offer functionality to terminate a registration. When a registration is terminated, the registrar shall discard the associated publication.
054	A registration shall support provision of publication information (i.e., metadata about the events published by the publisher that belongs to the registration and possibly also the event source).
055	A registration shall provide the consumer endpoint to which events shall be sent by the publisher.
056	A registration shall support provision of the publisher endpoint. The publisher is the entity that belongs to the registration and intends to send notifications to the consumer.
099	A registration shall provide the termination behavior defined for a resource with lifetime.

13.1.5 Brokered Publish Subscribe Requirements Package

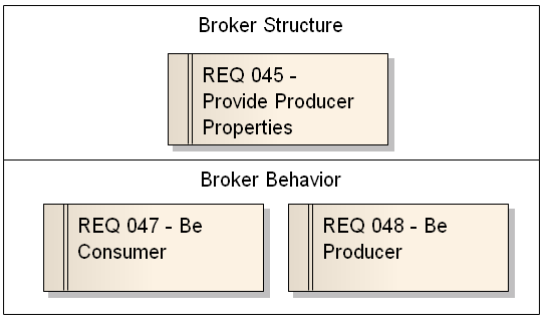

Figure 48: Brokered Publish Subscribe Requirements

Table 47: Brokered Publish Subscribe Requirements Details

Requirement Number	Requirement Normative Statement
045	A broker shall support the properties that a producer offers (resource properties, event channels, subscriptions, publications, service metadata).
047	A broker shall be a consumer, that is receive notifications from publishers.
048	A broker shall be a producer, that is realize provider as well as publisher and resource manager behavior. A broker shall match the messages it receives from publishers against existing subscriptions and publish matching messages to the delivery endpoint(s) defined for these subscriptions. It shall furthermore provide access to the resources it manages.

13.1.6 Registering Broker Requirements Package

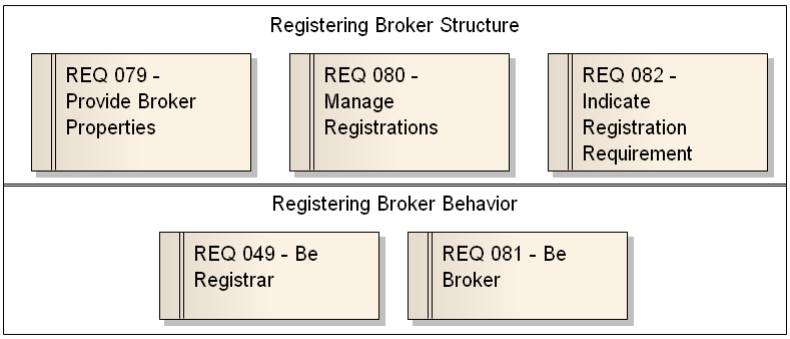

Figure 49: Registering Broker Requirements

Table 48: Registering Broker Requirements Details

Requirement Number	Requirement Normative Statement
049	A registering broker shall be a registrar, that is handle the registration of new publishers.
079	A registering broker shall provide the properties of a broker.
080	A registering broker shall provide information on the registrations it is currently managing.
081	A registering broker shall be a broker (receive incoming notifications and match them against existing subscriptions etc).
082	A registering broker - which requires publishers to register before they can send events to it - shall indicate this behavior via a policy statement.

13.1.7 Aggregation Channel Requirements Package

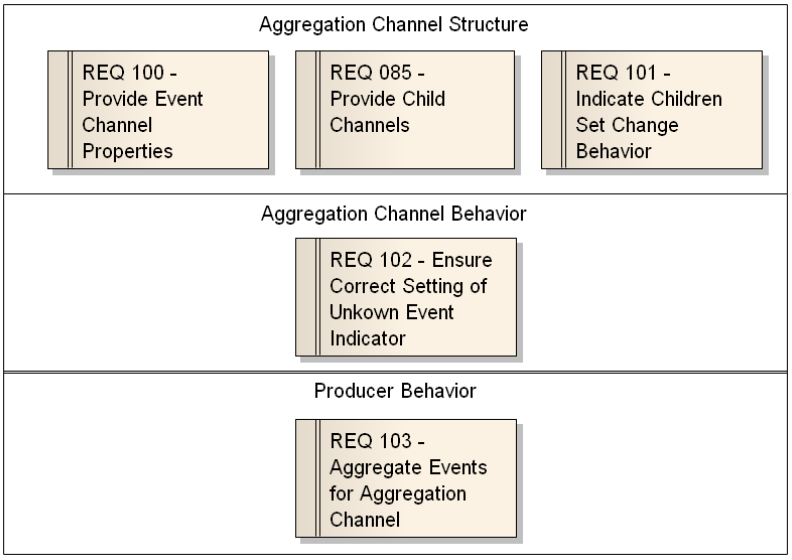

Figure 50: Aggregation Channel Requirements

Table 49: Aggregation Channel Requirements Details

Requirement Number	Requirement Normative Statement
085	An aggregation channel shall provide two or more child channels.
100	An aggregation event channel shall support the properties that an event channel offers (like information on published events).
101	An aggregation channel shall indicate if its set of children can or cannot change.
102	An aggregation shall indicate that unknown events can be published on it if one of its children does so.
103	A producer shall publish all events that are published on one of the child channels of an aggregation channel on that aggregation channel.

13.1.8 Ad Hoc Channel Requirements Package

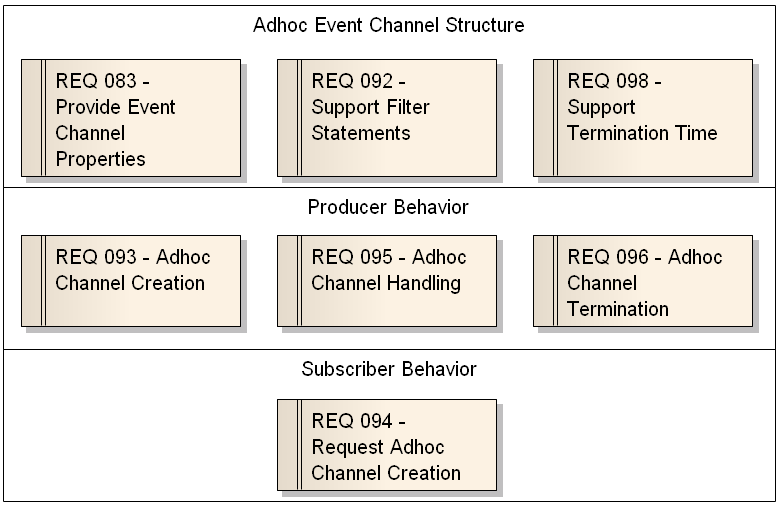

Figure 51: Ad Hoc Channel Requirements

Table 50: Ad Hoc Channel Requirements Details

Requirement Number	Requirement Normative Statement
083	An adhoc event channel shall support the properties that an event channel offers (like information on published events).
092	An adhoc event channel shall support the provision of filter criteria that were used in the subscription that caused the channel to be created.
093	A producer shall create an adhoc event channel if a proposed subscription indicates that events matching the subscription shall be posted on such a channel. The producer shall ensure that the termination time and filter statements of the channel are the same as in the subscription.
094	A subscriber shall request the creation of an adhoc channel by providing a value for the deliveryToChannel property in a proposed subscription, that is the AdHocChannel template, together with a name for the adhoc channel.
095	A producer shall publish all events that match the filter criteria of the adhoc event channel on the channel and allow subscriptions against it.

Requirement Number	Requirement Normative Statement
	The producer shall reset the termination time of the channel if the subscription that created it was renewed.
096	A producer shall terminate an adhoc channel when the subscription that created it is terminated.
098	An adhoc event channel shall support the provision of the termination time that applies to the subscription that caused the channel to be created.

13.1.9 Pausable Provider Requirements Package

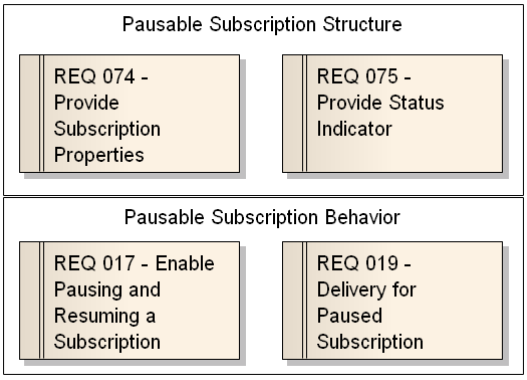

Figure 52: Pausable Provider Requirements

Table 51: Pausable Provider Requirements Details

Requirement Number	Requirement Normative Statement
017	A pauseable subscription shall provide operations to pause and resume it.
019	Events that match the filter criteria of a pausable subscription shall not be delivered to the subscription's deliveryTo endpoint while the subscription is paused.
074	A pausable subscription shall support the properties of a subscription.
075	A pausable subscription shall provide a boolean property that

Requirement Number	Requirement Normative Statement
	indicates if the subscription is paused or active.

13.1.10 Demand Based Publication Requirements Package

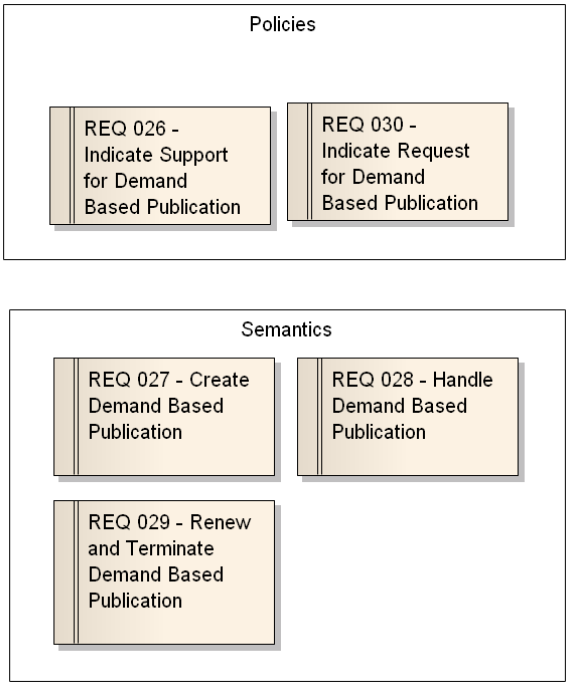

Figure 53: Demand Based Publication Requirements

Table 52: Demand Based Publication Requirements Details

Requirement Number	Requirement Normative Statement
026	A registrar that supports the demand based publication behavior shall indicate this through a policy statement.
027	A registrar that supports demand based publication shall - upon receiving a registration proposal that requests this behavior - create a pausable subscription at the publisher (that shall then act as a producer and support pausable subscriptions) whose endpoint shall be provided in the registration. The subscription shall target the events defined in the registration (either the ones in the given publication or all events) and shall initially be active. The registrar shall make the registration available and accept it once the producer

Requirement Number	Requirement Normative Statement
	accepted the subscription.
028	The registrar shall pause the subscription at the producer if it has no interest in events from it. It shall resume the subscription once interest is back.
029	The registrar shall renew / terminate the subscription when the registration is renewed / terminated.
030	An entity that requests demand based publication behavior in a registration shall indicate this through a policy statement.

14 Annex B – Publish Subscribe Requirements Realization Tables

This Annex contains the documentation of a given publish / subscribe technology with respect to how well it realizes the requirements listed in chapter 13 *Annex A - Publish Subscribe Requirements*.

The following grades were used to indicate how well a technology supports a given requirement:

1. Requirement is completely realized by technology
2. Requirement is partly realized by technology, missing functionality can be added
3. Required functionality is not foreseen by technology but can be added
4. Required functionality is not foreseen by technology and cannot be added

The following table shows how well WS-Notification supports each of the requirements. For each requirement a comment explains the realization grade choice. The summary of this investigation can be found in section 6.5.2.1.

Table 53: Realization of Requirements with WS-Notification

Requirement Number	Realization Grade	Comment
002	1	This is the Notify operation defined in WS-BaseNotification.
004	1	Each resource with lifetime can be terminated immediately using the WS-ResourceLifetime Destroy operation. In addition, WS-Notification defines additional operations for this purpose (Unsubscribe, DestroyRegistration).
006	1	The termination time of a resource with lifetime may be renewed via the SetTerminationTime operation defined by WS-ResourceLifetime. In addition, WS-Notification defines an explicit Renew operation for subscriptions.
007	1	Expression of termination time is supported through WS-ResourceLifetime TerminationTime resource property.
008	3	Although resource termination / destruction is considered in the WS-ResourceLifetime standard it does not define an actual termination event, only a channel and some properties the events posted on that channel must have. WS-Notification also does not define any termination message for subscriptions or registrations. The functionality can be added in many ways. On the one hand, an actual termination message format could be defined and an according event be

Requirement Number	Realization Grade	Comment
		published on the channel defined by WS-ResourceLifetime. On the other hand, a subscribe request can be extended to incorporate the endpoint where a termination event should be published to - this endpoint could then be included in the subscriptions' resource properties. The same mechanism can be applied to any WS-Resource.
009	1	A resource with lifetime (like a subscription) may execute scheduled termination behavior as defined by WS-ResourceLifetime.
010	3	WS-Notification does not support publication information and operations by itself. The missing functionality can be added, though. One option is to add a resource property to the NotificationProducer that lists its publications. Another option would be to let the entity that implements the NotificationProducer interface also be a ServiceGroup as defined by WS-ServiceGroup. In this group, the endpoints of all the publications would be listed (if they are implemented as WS-Resources).
011	3	As WS-Notification does not define publication resources, we can add the information we need if we define the publication resources ourselves (see REQ-076).
012	1	A NotificationProducer as defined by WS-Notification uses the Notify operation to publish events to consumers. This operation can also be used by other services (that do not implement the NotificationProducer interface) to publish events to a consumer.
013	3	Neither the Subscription resource nor the NotificationProducer interface define an operation that supports this functionality. However, an according resource property could be added to the resource that implements the NotificationProducer interface. The property could list the subscriptions. Another option would be to let the entity that implements the NotificationProducer interface also be a ServiceGroup as defined by WS-ServiceGroup. In this group the endpoints of all the Subscription(Manager)s would be listed.
014	1	This is achieved via the Subscribe operation implemented by a NotificationProducer.

Requirement Number	Realization Grade	Comment
015	1	This is the ConsumerReference in the Subscribe message / subscription resource as defined by WS-BaseNotification.
016	1	The NotificationProducer defined by WS-BaseNotification pushes matching notifications to the consumer of a subscription.
017	1	The PauseSubscription and ResumeSubscription messages contained in the Pausable Subscription Manager interface define the required functionality
019	2	This seems to be the standard behavior of a paused subscription as defined in WS-BaseNotification. However, the standard says that "in general, there is no way of knowing exactly when the pause will take effect". According constraints can be defined and made known through policy assertions.
020	3	Neither the RegisterPublisher nor the NotificationBroker interface define an operation that supports this functionality. However, an according resource property could be added to the resource that implements the RegisterPublisher interface. The property could list the registrations. Another option would be to let the entity that implements the RegisterPublisher interface also be a ServiceGroup as defined by WS-ServiceGroup. In this group the endpoints of all the PublisherRegistration(Manager)s would be listed.
021	1	WS-BrokeredNotification defines the RegisterPublisher interface for this purpose, with the RegisterPublisher operation.
023	2	WS-BrokeredNotification states that a publisher is only allowed to publish on the channels it is registered for. However, further requirements enforcing that the consumer to which a publisher is registered shall accept only those events that the publisher registered to send (this includes also the type of events and their encoding) are not stated - so these requirements would need to be added.
024	3	WS-BrokeredNotification defines the RequiresRegistration resource property that a NotificationBroker must include if it is exposed as a WS-Resource and supports the WS-ResourceProperties operations. However, this property is not defined for a Registrar in general - we would need to extend the applicability of the property so that it can be included in

Requirement Number	Realization Grade	Comment
		Resources that just implement the RegisterPublisher interface.
026	3	WS-BrokeredNotification does not say that a broker may reject a demand based publication (in other words: a RegisterPublisher message in which demand is set to true). This seems to indicate that a NotificationBroker always has to support demand based publication. However, a NotificationBroker may also reject a registration for a given reason - which sounds like it is allowed to reject a registration which asks for demand based publication behavior. In any case, no policy is defined to indicate that demand based publication is supported. This can be added, though.
027	1	WS-BrokeredNotification supports the semantics with its defined demand-based publication pattern.
028	1	WS-BrokeredNotification supports the semantics. "Interest in events" is considered to be similar to having a subscriber for a topic on which the demand based publication would publish events upon.
029	2	This is not explicitly defined in WS-BrokeredNotification. The subscription that belongs to such a publication seems to implicitly last as long as the registration. So this requirements seems to clarify the behavior of a NotificationBroker that implements demand based publishing.
030	1	This is the demand property of the RegisterPublisher message as defined in WS-BrokeredNotification.
033	2	Most of the functionality required by a provider is readily available (see REQ-014, REQ-013, REQ-016 and REQ-073).
034	2	A NotificationProducer uses the Notify operation to publish events to consumers. However, it does not provide an operation to retrieve publications. This can be added as outlined in REQ-010.
035	1	A NotificationProducer can offer access to the subscriptions and publications it manages as described in REQ-064.
037	2	A Subscription resource provides at least the Unsubscribe operation to immediately terminate it and the Renew operation to renew it - subscriptions in WS-Notification automatically support scheduled termination behavior. In

Requirement Number	Realization Grade	Comment
		addition, a Subscription resource may implement immediate and scheduled termination behavior as defined by WS-ResourceLifetime (which includes the renewal of the termination time via the SetTerminationTime operation defined by WS-ResourceLifetime). Only the termination event in case that a Subscription is terminated is not readily available - see REQ-067 - so would have to be added.
040	2	A WS-BaseNotification NotificationProducer has a WS-Addressing endpoint. It may have names as described for REQ-061. Documentation can be added as outlined in REQ-062 - same for policies (REQ-063).
041	3	A NotificationProducer does not provide a resource property or operation to access its subscriptions. However, an according resource property could be added to the resource that implements the NotificationProducer interface. The property could list the subscriptions. Another option would be to let the entity that implements the NotificationProducer interface also be a ServiceGroup as defined by WS-ServiceGroup. In this group, the endpoints of all the Subscription(Manager)s would be listed.
042	3	A NotificationProducer does not provide a resource property or operation to access its publications. However, an according resource property could be added to the resource that implements the NotificationProducer interface. The property could list the publications.
043	3	A NotificationProducer provides information on its channels via the TopicSet resource property. However, the channel information that is provided does not include all the information required for an event channel by itself (see REQ-084, REQ-086 and REQ-087). In addition, information about the events that are actually published and in which of channel it is published in which encoding is not provided - however, this can be added.
044	2	The WSDL document of a NotificationProducer provides information about implemented operations as well as policies. In addition the resoure properties of a NotificationProducer provide some more information on it. However, the WSDL description is not directly available from the service itself. This may change if WS-MetadataExchange is finalized. A NotificationProducer should provide capabilities. It can do so through a resource property or by realizing a new

Requirement Number	Realization Grade	Comment
		OGC interface that includes the GetCapabilities operation (at the moment each OWS unfortunately still needs to define his 'own' GetCapabilities operation as there is no global GetCapabilities element defined by OWS Common 1.1/1.2). Nevertheless, the missing metadata information can be added.
045	2	A NotificationBroker as defined by WS-BrokeredNotification is a NotificationProducer, that is a producer. The same statements as for a producer can thus be made (see REQ-040, REQ-041, REQ-042, REQ-043, REQ-044 and REQ-097). Overall, a NotificationProducer as defined by WS-Notification supports part of the information structure required for a producer - missing information can be added, though. This result therefore applies to a NotificationBroker as well.
047	1	As defined by WS-BrokeredNotification a NotificationBroker must implement the Consumer interface (defined in WS-BaseNotification) as well.
048	1	As defined by WS-BrokeredNotification a NotificationBroker must implement the NotificationProducer interface (defined in WS-BaseNotification) as well.
049	1	A NotificationBroker implements the RegisterPublisher interface.
052	2	A WS-BrokeredNotification PublisherRegistration has a WS-Addressing endpoint. It may have names as described for REQ-061. Documentation can be added as outlined in REQ-062 - same for policies (REQ-063). The PublisherRegistration supports a termination time through the InitialTerminationTime in the RegisterPublisher request and the wsrf-rl:TerminationTime resource property which can be added to the properties of the PublisherRegistration resource.
053	2	WS-BrokeredNotification defines the DestroyRegistration operation as part of the PubliserRegistrationManager interface. If we model the publication as a property belonging to the PublisherRegistration resource then it would automatically be destroyed as well. However, a further requirement in a WS-Notification realization of the Event Service functionality would be needed to detail and enforce the desired behavior.

Requirement Number	Realization Grade	Comment
054	3	WS-Notification does not define any specific resource properties in PublisherRegistration that provide metadata on published events. It can identify a set of channels where the publisher intends to publish events on - however, no specific information is given on the exact type of event. This information could be provided via new resource properties that an extension to the PublisherRegistration resource could define.
055	1	The ConsumerReference in the RegisterPublisherResponse provides the endpoint to which events shall be published. Note, however, that this endpoint is not part of the defined PublisherRegistration resource properties so would have to be added in case that this information should be queryable later on.
056	1	The PublisherReference in the PublisherRegistration resource can be used to provide the publisher endpoint as a WS-Addressing endpoint reference.
060	1	Each resource in WS-Notification is identified via a WS-Addressing endpoint.
061	2	A resource may have multiple WS-Addressing endpoints, in which the reference parameters could function as names. In addition, a WS-Resource may have any number of resource properties that provide its names.
062	3	Arbitrarily encoded and retrievable documentation of a WS-Resource like a WS-N subscription is not foreseen but may be added as WS-Resource properties.
063	3	There is no specific place in a WS-N resource (e.g. a subscription) that is dedicated to contain policy information. Any policy information may be added as WS-Resource property, though.
064	1	As the identifier of a resource in WS-Notification usually is a WS-Addressing endpoint, the WS-ResourceProperties operations can be used (on this endpoint) to access the properties of a resource. These properties may list the resources that are managed. In addition, a resource may provide ServiceGroups as defined by WS-ServiceGroup to provide access to the resources it manages.

Requirement Number	Realization Grade	Comment
067	3	Neither WSRF nor WS-Notification define an actual resource termination event. Only some termination event properties and a resource termination / destruction channel are defined. We can add the missing functionality by defining the termination event and extending the semantics of resource termination to emit this event accordingly.
069	1	The Notify request defined by WS-BaseNotification supports the inclusion of multiple NotificationMessage elements.
070	3	Although the Notify request in itself is unique, no specific identifier is added to it. This also applies to the NotificationMessages that are delivered with the request. However, the extension points in the Notify request can be used to add identity to the notification. Problem is that the NotificationMessage element itself does not provide extension points where the identity could be stored, so either a new element has to be designed which would be used internally at a consumer (although this could also be up to the consumer implementation if the Notify request itself contained an identifier) or the messages inside the NotificationMessage would need to have identity - the latter appears to be the most promising approach.
071	2	A WS-BaseNotification Subscription(Manager) resource has a WS-Addressing endpoint. It may have names as described for REQ-061. Documentation can be added as outlined in REQ-062 - same for policies (REQ-063, and an additional optional SubscriptionPolicy is available in Subscribe message and Subscription(Manager) resource). The Subscription(Manager) supports a termination time through the InitialTerminationTime in the Subscribe message and the wsrf-rl:TerminationTime resource property which can be added to the properties of the Subscription(Manager) resource.
072	1	This is the Filter property in a Subscribe message / Subscription(Manager) resource.
073	1	The Subscribe request defined in WS-BaseNotification allows filter statements in various languages to be included to define which events are of interest. Default behavior if no filter is provided is to send all events to the consumer.
074	2	In WS-BaseNotification, a subscription has the same properties regardless if it can be paused or not. The mapping

Requirement Number	Realization Grade	Comment
		grade was chosen as it is the same for REQ-071, which is not fully supported by WS-Notification out-of-the-box.
075	3	No resource property is defined by WS-Notification to indicate the status of a pausable subscription. This can be added.
076	3	WS-Notification does not define publication resources by itself. However, according resources can be added.
077	3	As WS-Notification does not define publication resources, we can add the information we need if we define the publication resources ourselves (see REQ-076).
078	3	WS-Notification does not say much on how a NotificationProducer can ensure that events are published on the right channels according to the channel definitions. It assumes that channel semantics exist and that the NotificationProducer follows these rules. However, we can define more strict requirements as needed.
079	2	See REQ-045, which is the only requirement that defined broker structure.
080	3	A NotificationBroker does not provide a resource property or operation to access its PublisherRegistrations. However, an according resource property could be added to the resource that implements the NotificationBroker interface. The property could list the PublisherRegistrations. Another option would be to let the entity that implements the NotificationBroker interface also be a ServiceGroup as defined by WS-ServiceGroup. In this group, the endpoints of all the PublisherRegistration(Manager)s would be listed.
081	1	This is standard behavior of a NotificationBroker (WS-BrokeredNotification does not differentiate between a broker and a registering broker).
082	1	This is the RequiresRegistration resource property of a NotificationBroker as defined in WS-BrokeredNotification.
083	3	WS-Notification does not define ad-hoc event channels. The required types and elements can be added.
084	3	Channels are provided in WS-Notification as as so called "topics" in a TopicSet - according to WS-Topics. The topics in a

Requirement Number	Realization Grade	Comment
		TopicSet or even TopicNamespace do not have explicit identity and also do not have a WS-Addressing endpoint to access them individually. All required properties could be added, however, as WS-Topics only defines very basic structure of a topic, leaving plenty of room for extending it.
085	3	see REQ-100
086	3	The Topic in a TopicNamespace may contain documentation of any type, so there event metadata could be inserted. However, WS-Notification does not define any element to document which events are published, and in which encoding they are published on a given channel / topic. This information can be added using the extension points defined in the various WS-Topics types.
087	2	WS-Topics defines that a Topic for which a list of message types is given can only publish events according to these types. So if the list is there then in that respect no unknown events are possible. However, this is not expressed as a boolean value.
088	3	WS-Topics does not define any element to document information about a given type of event. It only defines properties which could hold this information (like the Topic/documentation element). A new type / element can be defined and can then incorporate all properties required for an event metadata structure.
089	3	see REQ-088
090	3	see REQ-088
091	3	see REQ-088
092	3	see REQ-083
093	3	see REQ-095
094	3	WS-Notification does not define ad-hoc event channels. The Subscribe message can be extended to incorporate needed functionality and invoke required behavior.
095	3	WS-Notification does not define ad-hoc event channels. The required behavior can be defined.

Requirement Number	Realization Grade	Comment
096	3	see REQ-095
097	2	The TopicExpressionDialect resource property of a NotificationProducer is used to indicate which languages the producer supports for identifying channels (e.g. in subscriptions). Additional metadata on the filtering capabilities, like content filtering or complex event processing is not available. This can be added through an according capabilities section as described in REQ-044.
098	3	see REQ-083
099	2	A PublisherRegistration provides at least the DestroyRegistration operation to immediately terminate it. In addition, a PublisherRegistration resource may implement immediate and scheduled termination behavior defined by WS-ResourceLifetime (which includes the renewal of the termination time via the SetTerminationTime operation defined by WS-ResourceLifetime). Only the termination event in case that a PublisherRegistration is terminated is not readily available - see REQ-067 - so would have to be added.
100	3	WS-Notification does not define aggregation channels. The required types and elements can be added.
101	3	see REQ-100
102	3	WS-Notification does not define aggregation channels. The required behavior can be defined.
103	3	see REQ-102
104	3	WS-Topics defines the TopicNamespace and TopicSet elements to contain information about channels and - to a limited extent - the events published on them. A TopicSet does not have explicit identity through a WS-Addressing endpoint. However, it can be requested from a NotificationProducer as it can be modelled as one of its resource properties. A TopicNamespace has identity because of its targetNamespace attribute. In addition a TopicNamespace may have a name. A NotificationProducer may provide a handle to retrieve TopicNamespace information, by adding the wstop:TopicNamespaceLocation attribute in TopicExpressions ... or even entries in a TopicSet. However, this is not sufficiently documented in the WS-Topics standard. A TopicNamespace does provide an optional

Requirement Number	Realization Grade	Comment
		property to contain documentation of any type. To summarize, WS-Topics defines some elements to document eventing information but does not provide it in a sufficient way. Missing information can be added, however, using various extension points.
105	2	This is done via the TopicSet of a service. However, full information of a channel as defined in REQ-84, REQ-086 and REQ-089 is not readily available. This can be added, though, using the extension points available in a TopicSet.
106	3	WS-Notification only provides extension points (e.g. in a TopicNamespace and TopicSet) to contain information on published events. They can be used to provide the required information (for which new types / xml elements need to be defined).
107	2	In a TopicNamespace as defined by WS-Topics, a topic may list the types of messages that can be published on the topic. This provides information on the event encoding. However, it does not suffice to determine exactly which event type is published in which encoding on the channel (multiple event types may be published using the same encoding on the same channel - they can then for example be differentiated using a code that is contained in the event encoding).

15 Annex C – Additional Event Discovery Material

15.1 CSW-ebRIM Extension Package for Event Service Discovery

The following is the XML representation of the Extension Package for Event Service Discovery which should be compatible with any CSW-ebRIM v1.0.1 registry service. This extension package is presented in the form of an **INSERT** transaction to facilitate loading into a registry.

Listing 7: Insert Transaction request to populate registry with Event Service Discovery Extension Package

```
<?xml version="1.0" encoding="UTF-8"?>
<Transaction service="CSW" version="2.0.2"
  xmlns="http://www.opengis.net/cat/csw/2.0.2"
  xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0">
  <Insert>

    <rim:RegistryPackage  id="urn:ogc:def:package:ows7-event:discovery">
      <rim:Name>
        <rim:LocalizedString xml:lang="en" value="OWS-7 Eventing
Discovery model package" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString xml:lang="en" value="Provides objectTypes,
associationTypes and classification
          schemes for Eventing Discovery model used in OWS-7." />
      </rim:Description>

      <rim:RegistryObjectList>

        <!--
          link this Extension Package to the "Root" Extension Package
          ~ this defines this RegistryPackage object to be an "Extension
Package".
        -->
        <rim:Association id="assoc-root-pkg-01"
          associationType="urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasMember"
          sourceObject="urn:ogc:def:ebRIM-RegistryPackage:OGC:Root"
          targetObject="urn:ogc:def:package:ows7-event:discovery"/>

        <!-- ObjectTypes -->
        <rim:ClassificationNode id="urn:ogc:def:rim-object-
type:ows7:EventChannel"
          objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
          parent="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ExtrinsicObject:XML"
          code="EventChannel">
          <rim:Name>
            <rim:LocalizedString xml:lang="en" value="EventChannel" />
          </rim:Name>
          <rim:Description>
```

```
            <rim:LocalizedString xml:lang="en" value="An EventChannel is
a logical
              stream of events which may consist of one or more distinct
              EventTypes." />
          </rim:Description>
        </rim:ClassificationNode>
        <rim:ClassificationNode id="urn:ogc:def:rim-object-
type:ows7:EventType"
          objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
          parent="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ExtrinsicObject:XML"
          code="EventType">
          <rim:Name>
            <rim:LocalizedString xml:lang="en" value="EventType" />
          </rim:Name>
          <rim:Description>
            <rim:LocalizedString xml:lang="en" value="An EventType is a
logical set
              of event properties.  EventTypes may be bound to an
EventChannel with
              an explicit encoding." />
          </rim:Description>
        </rim:ClassificationNode>

        <!-- AssociationTypes -->
        <rim:ClassificationNode
          id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasChannel"
          objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
          parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
          code="HasChannel">
          <rim:Name>
            <rim:LocalizedString xml:lang="en" value="HasChannel" />
          </rim:Name>
          <rim:Description>
            <rim:LocalizedString xml:lang="en" value="Eventing Service
metadata
              records are linked with EventChannel instances via a
HasChannel
              association." />
          </rim:Description>
        </rim:ClassificationNode>
        <rim:ClassificationNode
          id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:ChannelBinding"
          objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
          parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
          code="ChannelBinding">
          <rim:Name>
            <rim:LocalizedString xml:lang="en" value="ChannelBinding" />
          </rim:Name>
          <rim:Description>
```



```
            <rim:LocalizedString xml:lang="en" value="EventChannel
instances are
              linked with EventType instances via the ChannelBinding
association."
             />
          </rim:Description>
        </rim:ClassificationNode>
        <rim:ClassificationNode
          id="urn:oasis:names:tc:ebxml-
regrep:AssociationType:ConstrainedBy"
          objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
          parent="urn:oasis:names:tc:ebxml-
regrep:classificationScheme:AssociationType"
          code="ConstrainedBy">
          <rim:Name>
            <rim:LocalizedString xml:lang="en" value="ConstrainedBy" />
          </rim:Name>
          <rim:Description>
            <rim:LocalizedString xml:lang="en" value="An
ExtrinsicObject's
              repository item is &amp;quot;ConstrainedBy&amp;quot; an abstract
              schema. Schema may be defined in any constraint language."
/>
          </rim:Description>
        </rim:ClassificationNode>

        <!-- new Service Type Classification nodes -->
        <rim:ClassificationNode
          id="urn:ogc:serviceType:SensorEventService:0.3"
          parent="urn:ogc:def:ebRIM-ClassificationScheme:ISO-
19119:2005:Services:Subscription"
          code="SES">
          <rim:Name>
            <rim:LocalizedString xml:lang="en" value="Sensor Event
Service (SES)" />
          </rim:Name>
          <rim:Description>
            <rim:LocalizedString xml:lang="en" value="OGC 08-133
Discussion Paper" />
          </rim:Description>
        </rim:ClassificationNode>
        <rim:ClassificationNode
          id="urn:ogc:serviceType:AimEventService:0.1"
          parent="urn:ogc:def:ebRIM-ClassificationScheme:ISO-
19119:2005:Services:Subscription"
          code="AIES">
          <rim:Name>
            <rim:LocalizedString xml:lang="en" value="Aeronautical
Information Event Service (AIES)" />
          </rim:Name>
          <rim:Description>
            <rim:LocalizedString xml:lang="en" value="The Aeronautical
Information Exchange Model (AIXM)
              is designed to enable the management and distribution of
Aeronautical Information Services
              (AIS) data in digital format.  See
[http://www.aixm.aero]." />
```

```
          </rim:Description>
        </rim:ClassificationNode>

        <!--
          Filter Encoding 2.0.0 scheme
        -->
        <rim:ClassificationScheme
          xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
          xmlns:wrs="http://www.opengis.net/cat/wrs/1.0"
          xmlns:xlink="http://www.w3.org/1999/xlink"
          id="urn:x-ogc:def:rim-scheme:fes-2.0-ops"
          objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationScheme"
          nodeType="urn:oasis:names:tc:ebxml-regrep:NodeType:UniqueCode"
          isInternal="true">

          <!--
             Scheme for assigning OGC Filter (Filter Encoding 2.0.0)
operators to RegistryObjects
             (typically Services) to indicate which Filter features are
supported by a particular
             Service instance.

             These Filter capabilities are identified, in detail, within
this scheme:
                Logical operators
                Comparison operators
                Spatial operators
                Geometry operands
                Temporal operators

             Aspects of Filter capabilties which have not yet been
included in this scheme:
                Conformance constraints
                Id Capabilities
                Functions
                ExtendedCapabilities
          -->

          <rim:Name>
            <rim:LocalizedString xml:lang="en" value="Filter Encoding
Specification (FES) 2.0.0 Operators" />
          </rim:Name>
          <rim:Description>
            <rim:LocalizedString xml:lang="en" value="All FES 2.0.0
operators, including generic extensions, are
                represented in this scheme.  It is intended for
classifying OGC Web service metadata to indicate which FES operators the
service supports." />
          </rim:Description>

          <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-2.0-
ops:logical"
            objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
            parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops"
            code="logical-ops">
            <rim:Name>
```



```
              <rim:LocalizedString xml:lang="en" value="Logical
Operators" />
            </rim:Name>
            <rim:Description>
              <rim:LocalizedString xml:lang="en" value="Logical
Operators include: AND, OR and NOT." />
            </rim:Description>
          </rim:ClassificationNode>

          <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-2.0-
ops:comparison"
            objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
            parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops"
            code="comparison-ops">
            <rim:Name>
              <rim:LocalizedString xml:lang="en" value="Comparison
Operators" />
            </rim:Name>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:comparison:equal"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:comparison"
              code="prop-equal">
              <rim:Name>
                <rim:LocalizedString xml:lang="en"
value="PropertyIsEqualTo operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:comparison:not-equal"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:comparison"
              code="prop-not-equal">
              <rim:Name>
                <rim:LocalizedString xml:lang="en"
value="PropertyIsNotEqualTo operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:comparison:lt"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:comparison"
              code="prop-lt">
              <rim:Name>
                <rim:LocalizedString xml:lang="en"
value="PropertyIsLessThan operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:comparison:gt"
```



```
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:comparison"
              code="prop-gt">
              <rim:Name>
                <rim:LocalizedString xml:lang="en"
value="PropertyIsGreaterThan operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:comparison:lte"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:comparison"
              code="prop-lte">
              <rim:Name>
                <rim:LocalizedString xml:lang="en"
value="PropertyIsLessThanOrEqualTo operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:comparison:gte"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:comparison"
              code="prop-gte">
              <rim:Name>
                <rim:LocalizedString xml:lang="en"
value="PropertyIsGreaterThanOrEqualTo operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:comparison:like"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:comparison"
              code="prop-like">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="PropertyIsLike
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:comparison:null"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:comparison"
              code="prop-null">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="PropertyIsNull
operator" />
              </rim:Name>
            </rim:ClassificationNode>
```



```
            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:comparison:nil"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:comparison"
              code="prop-nil">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="PropertyIsNil
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:comparison:between"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:comparison"
              code="prop-between">
              <rim:Name>
                <rim:LocalizedString xml:lang="en"
value="PropertyIsBetween operator" />
              </rim:Name>
            </rim:ClassificationNode>

          </rim:ClassificationNode>

          <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-2.0-
ops:spatial-cap"
            objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
            parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops"
            code="spatial-cap">
            <rim:Name>
              <rim:LocalizedString xml:lang="en" value="Spatial
Capabilities" />
            </rim:Name>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:geometry-operands"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial-cap"
              code="geometry-operands">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="Geometry
Operands" />
              </rim:Name>

              <!-- from GML Simple Features geometry types -->
              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:geometry-operands:gml-point"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:geometry-
operands"
                code="gml-point">
                <rim:Name>
```

```
                  <rim:LocalizedString xml:lang="en" value="Point
geometry" />
                </rim:Name>
              </rim:ClassificationNode>
              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:geometry-operands:gml-linestring"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:geometry-
operands"
                code="gml-linestring">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="LineString
geometry" />
                </rim:Name>
              </rim:ClassificationNode>
              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:geometry-operands:gml-polygon"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:geometry-
operands"
                code="gml-polygon">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="Polygon
geometry" />
                </rim:Name>
              </rim:ClassificationNode>
              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:geometry-operands:gml-envelope"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:geometry-
operands"
                code="gml-envelope">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="GML
Envelope" />
                </rim:Name>
              </rim:ClassificationNode>

            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial-cap"
              code="spatial-ops">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="Spatial
Operators" />
              </rim:Name>

              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial:bbox"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
```



```
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial"
                code="bbox-op">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="BBOX
operator" />
                </rim:Name>
              </rim:ClassificationNode>

              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial:equals"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial"
                code="equals-op">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="Equals
operator" />
                </rim:Name>
              </rim:ClassificationNode>

              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial:disjoint"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial"
                code="disjoint-op">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="Disjoint
operator" />
                </rim:Name>
              </rim:ClassificationNode>

              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial:intersects"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial"
                code="intersects-op">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="Intersects
operator" />
                </rim:Name>
              </rim:ClassificationNode>

              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial:touches"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial"
                code="touches-op">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="Touches
operator" />
                </rim:Name>
              </rim:ClassificationNode>

              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial:crosses"
```

```
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial"
                code="crosses-op">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="Crosses
operator" />
                </rim:Name>
              </rim:ClassificationNode>

              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial:within"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial"
                code="within-op">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="Within
operator" />
                </rim:Name>
              </rim:ClassificationNode>

              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial:contains"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial"
                code="contains-op">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="Contains
operator" />
                </rim:Name>
              </rim:ClassificationNode>

              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial:overlaps"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial"
                code="overlaps-op">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="Overlaps
operator" />
                </rim:Name>
              </rim:ClassificationNode>

              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial:beyond"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial"
                code="beyond-op">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="Beyond
operator" />
                </rim:Name>
              </rim:ClassificationNode>
```



```
              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial:dwithin"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial"
                code="dwithin-op">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="DWithin
operator" />
                </rim:Name>
              </rim:ClassificationNode>

              <!-- plus:  Extended Spatial Operator  -->
              <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:spatial:ext"
                objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
                parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:spatial"
                code="ext-spatial-op">
                <rim:Name>
                  <rim:LocalizedString xml:lang="en" value="Extended
spatial operator" />
                </rim:Name>
              </rim:ClassificationNode>

            </rim:ClassificationNode>
          </rim:ClassificationNode>

          <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-2.0-
ops:temporal"
            objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
            parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops"
            code="temporal">
            <rim:Name>
              <rim:LocalizedString xml:lang="en" value="Temporal
Operators" />
            </rim:Name>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:after"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="after-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="After
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:before"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="before-op">
              <rim:Name>
```

```
                <rim:LocalizedString xml:lang="en" value="Before
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:begins"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="begins-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="Begins
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:begunby"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="begunby-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="BegunBy
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:tcontains"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="tcontains-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="TContains
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:during"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="during-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="During
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:tequals"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
```



```
              code="tequals-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="TEquals
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:toverlaps"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="toverlaps-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="TOverlaps
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:meets"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="meets-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="Meets
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:overlappedby"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="overlappedby-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="OverlappedBy
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:metby"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="metby-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="MetBy
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:ends"
```

```
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="ends-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="Ends operator"
/>
              </rim:Name>
            </rim:ClassificationNode>

            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:endedby"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="endedby-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="EndedBy
operator" />
              </rim:Name>
            </rim:ClassificationNode>

            <!-- plus:  Extended Temporal Operator  -->
            <rim:ClassificationNode id="urn:x-ogc:def:rim-scheme:fes-
2.0-ops:temporal:ext"
              objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ClassificationNode"
              parent="urn:x-ogc:def:rim-scheme:fes-2.0-ops:temporal"
              code="ext-temporal-op">
              <rim:Name>
                <rim:LocalizedString xml:lang="en" value="Extended
temporal operator" />
              </rim:Name>
            </rim:ClassificationNode>
          </rim:ClassificationNode>
        </rim:ClassificationScheme>
      </rim:RegistryObjectList>
    </rim:RegistryPackage>
  </Insert>
</Transaction>
```

15.2 Additional Sample Queries for Event Service Discovery

The queries below can be pasted into the OWS-7 demonstration Query utility page here:

http://registry.galdosinc.com/ows7/util/query

This query page requires authentication --- use these credentials: ows7event / 0pen4me

Note: The password contains two numbers, a zero (0) and a four (4).

Listing 8: Query #1 - Find SFE event services which offer an event channel for DetectedChanges

```
<?xml version="1.0" encoding="UTF-8"?>
<GetRecords xmlns="http://www.opengis.net/cat/csw/2.0.2"
  xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:wrs="http://www.opengis.net/cat/wrs/1.0"
  outputFormat="application/xml; charset=UTF-8"
  service="CSW" version="2.0.2"
  startPosition="1" maxRecords="10"
  resultType="results">
  <!--
    Find SFE event services which offer an event channel for
"DetectedChanges".

    This query exploits the "HasChannel" association and the fact that
the channel-type
    is a fixed string held in the Name property of the EventChannel
registry-object.
  -->
  <Query typeNames="Service Classification Association ExtrinsicObject">
    <ElementSetName typeNames="Service">full</ElementSetName>
    <Constraint version='1.1.0'>
      <ogc:Filter>
        <ogc:And>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>

<ogc:PropertyName>Classification/@classifiedObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>Classification/@classificationNode</ogc:PropertyName>

<ogc:Literal>urn:ogc:serviceType:SensorEventService:0.3</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>

<ogc:PropertyName>Association/@sourceObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>Association/@associationType</ogc:PropertyName>
            <ogc:Literal>urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasChannel</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>ExtrinsicObject/@id</ogc:PropertyName>

<ogc:PropertyName>Association/@targetObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>ExtrinsicObject/@objectType</ogc:PropertyName>
            <ogc:Literal>urn:ogc:def:rim-object-
type:ows7:EventChannel</ogc:Literal>
          </ogc:PropertyIsEqualTo>
```

```
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>ExtrinsicObject/Name/LocalizedString/@value</ogc:Prope
rtyName>
            <ogc:Literal>DetectedChanges</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:And>
      </ogc:Filter>
    </Constraint>
  </Query>
</GetRecords>
```

Listing 9: Query #2 - Find SFE event services which support the spatial operator BBOX

```
<?xml version="1.0" encoding="UTF-8"?>
<GetRecords xmlns="http://www.opengis.net/cat/csw/2.0.2"
  xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:wrs="http://www.opengis.net/cat/wrs/1.0"
  outputFormat="application/xml; charset=UTF-8"
  service="CSW" version="2.0.2"
  startPosition="1" maxRecords="10"
  resultType="results">
  <!--
    Find SFE event services which support the spatial operator BBOX.
  -->
  <Query typeNames="Service Classification_c1_c2">
    <ElementSetName typeNames="Service">full</ElementSetName>
    <Constraint version='1.1.0'>
      <ogc:Filter>
        <ogc:And>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>
            <ogc:PropertyName>$c1/@classifiedObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>$c1/@classificationNode</ogc:PropertyName>

<ogc:Literal>urn:ogc:serviceType:SensorEventService:0.3</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>
            <ogc:PropertyName>$c2/@classifiedObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>$c2/@classificationNode</ogc:PropertyName>
            <ogc:Literal>urn:x-ogc:def:rim-scheme:fes-2.0-
ops:spatial:bbox</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:And>
      </ogc:Filter>
    </Constraint>
  </Query>
</GetRecords>
```


Listing 10: Query #3 - Find AIM event services which offer an event channel for "Notams"

```
<?xml version="1.0" encoding="UTF-8"?>
<GetRecords xmlns="http://www.opengis.net/cat/csw/2.0.2"
  xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:wrs="http://www.opengis.net/cat/wrs/1.0"
  outputFormat="application/xml; charset=UTF-8"
  service="CSW" version="2.0.2"
  startPosition="1" maxRecords="10"
  resultType="results">
  <!--
    Find AIM event services which offer an event channel for "Notams".

    This query exploits the "HasChannel" association and the fact that
the channel-type
    is a fixed string held in the Name property of the EventChannel
registry-object.
  -->
  <Query typeNames="Service Classification Association ExtrinsicObject">
    <ElementSetName typeNames="Service">full</ElementSetName>
    <Constraint version='1.1.0'>
      <ogc:Filter>
        <ogc:And>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>

<ogc:PropertyName>Classification/@classifiedObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>Classification/@classificationNode</ogc:PropertyName>

<ogc:Literal>urn:ogc:serviceType:AimEventService:0.1</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>

<ogc:PropertyName>Association/@sourceObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>Association/@associationType</ogc:PropertyName>
            <ogc:Literal>urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasChannel</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>ExtrinsicObject/@id</ogc:PropertyName>

<ogc:PropertyName>Association/@targetObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>ExtrinsicObject/@objectType</ogc:PropertyName>
            <ogc:Literal>urn:ogc:def:rim-object-
type:ows7:EventChannel</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
```



```
<ogc:PropertyName>ExtrinsicObject/Name/LocalizedString/@value</ogc:Prope
rtyName>
            <ogc:Literal>Notams</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:And>
      </ogc:Filter>
    </Constraint>
  </Query>
</GetRecords>
```

Listing 11: Query #4 - Find AIM event services which support the temporal operators (at least one)

```
<?xml version="1.0" encoding="UTF-8"?>
<GetRecords xmlns="http://www.opengis.net/cat/csw/2.0.2"
  xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:wrs="http://www.opengis.net/cat/wrs/1.0"
  outputFormat="application/xml; charset=UTF-8"
  service="CSW" version="2.0.2"
  startPosition="1" maxRecords="10"
  resultType="results">
  <!--
    Find AIM event services which support the temporal operators (at
least one).
  -->
  <Query typeNames="Service Classification_c1_c2">
    <ElementSetName typeNames="Service">full</ElementSetName>
    <Constraint version='1.1.0'>
      <ogc:Filter>
        <ogc:And>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>
            <ogc:PropertyName>$c1/@classifiedObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>$c1/@classificationNode</ogc:PropertyName>

<ogc:Literal>urn:ogc:serviceType:AimEventService:0.1</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>
            <ogc:PropertyName>$c2/@classifiedObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>$c2/@classificationNode</ogc:PropertyName>
            <ogc:Literal>urn:x-ogc:def:rim-scheme:fes-2.0-
ops:temporal</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:And>
      </ogc:Filter>
    </Constraint>
  </Query>
</GetRecords>
```


Listing 12: Query #5 - Find AIM event services with "Weather" channel and support for GML Point operands (for spatial operators)

```
<?xml version="1.0" encoding="UTF-8"?>
<GetRecords xmlns="http://www.opengis.net/cat/csw/2.0.2"
  xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:wrs="http://www.opengis.net/cat/wrs/1.0"
  outputFormat="application/xml; charset=UTF-8"
  service="CSW" version="2.0.2"
  startPosition="1" maxRecords="10"
  resultType="results">
  <!--
    Find AIM event services which offers a channel for "Weather" and
    supports GML Point operands (for spatial operators).
  -->
  <Query typeNames="Service Classification_c1_c2 Association
ExtrinsicObject">
    <ElementSetName typeNames="Service">full</ElementSetName>
    <Constraint version='1.1.0'>
      <ogc:Filter>
        <ogc:And>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>
            <ogc:PropertyName>$c1/@classifiedObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>$c1/@classificationNode</ogc:PropertyName>

<ogc:Literal>urn:ogc:serviceType:AimEventService:0.1</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>
            <ogc:PropertyName>$c2/@classifiedObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>$c2/@classificationNode</ogc:PropertyName>
            <ogc:Literal>urn:x-ogc:def:rim-scheme:fes-2.0-ops:geometry-
operands:gml-point</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>Service/@id</ogc:PropertyName>

<ogc:PropertyName>Association/@sourceObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>Association/@associationType</ogc:PropertyName>
            <ogc:Literal>urn:oasis:names:tc:ebxml-
regrep:AssociationType:HasChannel</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>ExtrinsicObject/@id</ogc:PropertyName>

<ogc:PropertyName>Association/@targetObject</ogc:PropertyName>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
```

```
<ogc:PropertyName>ExtrinsicObject/@objectType</ogc:PropertyName>
            <ogc:Literal>urn:ogc:def:rim-object-
type:ows7:EventChannel</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>

<ogc:PropertyName>ExtrinsicObject/Name/LocalizedString/@value</ogc:Prope
rtyName>
            <ogc:Literal>Weather</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:And>
      </ogc:Filter>
    </Constraint>
  </Query>
</GetRecords>
```


16 Annex D – Event Metadata XML Implementation

16.1 XML Schema for Event Metadata

The following schema was implements the UML model of the event metadata package – see section 6.2.1.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:es="http://www.opengis.net/es/0.1"
targetNamespace="http://www.opengis.net/es/0.1" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <annotation>
    <documentation>This package defines types / classes that capture eventing
metadata.</documentation>
  </annotation>
  <import namespace="http://www.opengis.net/gml/3.2"
schemaLocation="http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>
  <element name="EventChannelRelationship" type="es:EventChannelRelationshipType"
substitutionGroup="gml:AbstractGML">
    <annotation>
      <documentation>Represents the association class of the association that exists
between an EventChannel and EventMetadata.</documentation>
    </annotation>
  </element>
  <complexType name="EventChannelRelationshipType">
    <complexContent>
      <extension base="gml:AbstractGMLType">
        <sequence>
          <element name="eventEncoding" nillable="true" maxOccurs="unbounded">
            <annotation>
              <documentation>Encoding of an event for a given channel.

Note: the property may be null in cases in which the exact encoding of an event cannot be
determined. This should be indicated using the nilReason attribute.</documentation>
            </annotation>
            <complexType>
              <complexContent>
                <extension base="gml:CodeType">
                  <attribute name="nilReason" type="gml:NilReasonType"/>
                </extension>
              </complexContent>
            </complexType>
          </element>
          <element name="targetEvent" type="es:EventMetadataPropertyType"/>
          <element name="targetChannel" type="es:EventChannelPropertyType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="EventChannelRelationshipPropertyType">
    <sequence minOccurs="0">
      <element ref="es:EventChannelRelationship"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
  <element name="EventingInformation" type="es:EventingInformationType"
substitutionGroup="gml:AbstractGML">
    <annotation>
      <documentation>This type serves as the container to describe EventChannel /
EventMetadata and their relationships.</documentation>
    </annotation>
  </element>
  <complexType name="EventingInformationType">
    <complexContent>
      <extension base="gml:AbstractGMLType">
```

```
        <sequence>
          <element name="channel" type="es:EventChannelPropertyType" minOccurs="0"
maxOccurs="unbounded">
            <annotation>
              <documentation>A channel supported by the event service.</documentation>
            </annotation>
          </element>
          <element name="event" type="es:EventMetadataPropertyType" minOccurs="0"
maxOccurs="unbounded">
            <annotation>
              <documentation>Metadata about a type of event that is published by the
service.</documentation>
            </annotation>
          </element>
          <element name="relationship" type="es:EventChannelRelationshipPropertyType"
minOccurs="0" maxOccurs="unbounded">
            <annotation>
              <documentation>The relationship that explains the encoding being used to
publish a given type of event on a given channel.</documentation>
            </annotation>
          </element>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="EventingInformationPropertyType">
    <sequence minOccurs="0">
      <element ref="es:EventingInformation"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
  <element name="AdhocEventChannel" type="es:AdhocEventChannelType"
substitutionGroup="es:EventChannel">
    <annotation>
      <documentation>Represents an &lt;i&gt;EventChannel &lt;/i&gt;that was created ad
hoc as additional delivery target for a subscription.

Other subscribers can explore existing ad hoc channels and which events they target (via
the filter statements and documentation) and subscribe for the same events. Handling
subscriptions that target the same events is thus made easier for both subscribers and
the producer.

Subscribers to this kind of channel should be aware that the channel may terminate any
time (because the underlying subscription may be terminated at any time). A given
termination time is therefore only indicative.</documentation>
    </annotation>
  </element>
  <complexType name="AdhocEventChannelType">
    <complexContent>
      <extension base="es:EventChannelType">
        <sequence>
          <element name="filter" type="anyType" minOccurs="0" maxOccurs="unbounded">
            <annotation>
              <documentation>Filter statement used by the subscription that created the
ad-hoc channel.</documentation>
            </annotation>
          </element>
          <element name="terminationTime" type="gml:TimeInstantPropertyType"
minOccurs="0">
            <annotation>
              <documentation>Time when the channel is set to automatically expire. This
shall be the same as the termination time of the subscription that created the ad hoc
channel. Note that the termination time - if given - is only indicative, as the
underlying subscription may be terminated at any time.</documentation>
            </annotation>
          </element>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="AdhocEventChannelPropertyType">
```



```
    <sequence minOccurs="0">
      <element ref="es:AdhocEventChannel"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
  <element name="EventMetadata" type="es:EventMetadataType"
substitutionGroup="gml:AbstractGML">
    <annotation>
      <documentation>Represents semantics of an event.</documentation>
    </annotation>
  </element>
  <complexType name="EventMetadataType">
    <complexContent>
      <extension base="gml:AbstractGMLType">
        <sequence>
          <element name="definition" type="anyType">
            <annotation>
              <documentation>Defines the event in as much detail as possible. Any type of
definition is suitable, from a pure textual to a pure ontology based
definition.</documentation>
            </annotation>
          </element>
          <element name="eventName" type="gml:CodeType" minOccurs="0"
maxOccurs="unbounded">
            <annotation>
              <documentation>Name of the event - rather, the happening it represents -
assigned by a given domain.</documentation>
            </annotation>
          </element>
          <element name="publishedOn" type="es:EventChannelRelationshipPropertyType"
maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="EventMetadataPropertyType">
    <sequence minOccurs="0">
      <element ref="es:EventMetadata"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
  <element name="AggregationChannel" type="es:AggregationChannelType"
substitutionGroup="es:EventChannel">
    <annotation>
      <documentation>Represents a channel in which all events from the child channels are
published as well. This makes it easier for clients to subscribe for a number of
channels.</documentation>
    </annotation>
  </element>
  <complexType name="AggregationChannelType">
    <complexContent>
      <extension base="es:EventChannelType">
        <sequence>
          <element name="final" type="boolean">
            <annotation>
              <documentation>true if the aggregation channel's set of child channels does
not change (children are removed or added) - false otherwise</documentation>
            </annotation>
          </element>
          <element name="child" type="es:EventChannelPropertyType" minOccurs="2"
maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="AggregationChannelPropertyType">
    <sequence minOccurs="0">
      <element ref="es:AggregationChannel"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
```

```
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
  <element name="EventChannel" type="es:EventChannelType"
substitutionGroup="gml:AbstractGML">
    <annotation>
      <documentation>An EventChannel is a Resource. Multiple events may be published on
the channel. In some cases it may not be possible to identify these
events.</documentation>
    </annotation>
  </element>
  <complexType name="EventChannelType">
    <complexContent>
      <extension base="gml:AbstractGMLType">
        <sequence>
          <element name="unknownEventsPossible" type="boolean" default="true">
            <annotation>
              <documentation>True if more than the listed events may be published on this
channel, otherwise false.</documentation>
            </annotation>
          </element>
          <element name="publishedEvent" type="es:EventChannelRelationshipPropertyType"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="EventChannelPropertyType">
    <sequence minOccurs="0">
      <element ref="es:EventChannel"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
</schema>
```

16.2 Event Metadata for Aviation Service – XML Example

The following listing contains an example of an Aviation service's event metadata contained as extension in a WS-Notification TopicSet. Such a TopicSet provides rudimentary information about the event channels supported by a service.

Listing 13: Example of Extended TopicSet with Aviation Service Event Metadata

```
<wstop:TopicSet xmlns:wstop="http://docs.oasis-open.org/wsn/t-1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/wsn/t-1 http://docs.oasis-
open.org/wsn/t-1.xsd" xmlns:es="http://www.opengis.net/es/0.1"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:aimtns="http://www.opengis.net/ows7/aim/channels">
  <aimtns:Notams wstop:topic="true"/>
  <aimtns:Weather wstop:topic="true"/>
  <!-- the eventing information -->
  <es:EventingInformation gml:id="aimEventingInformation"
xsi:schemaLocation="http://www.opengis.net/es/0.1 ../es.xsd">
    <es:channel>
      <es:EventChannel gml:id="notamsChannel">
        <gml:name
codeSpace="http://www.opengis.net/ows7/aim/channels">Notams</gml:name>
        <es:unknownEventsPossible>false</es:unknownEventsPossible>
        <es:publishedEvent xlink:href="#notamsRel1"/>
      </es:EventChannel>
    </es:channel>
    <es:channel>
```



```
      <es:EventChannel gml:id="weatherChannel">
        <gml:name
codeSpace="http://www.opengis.net/ows7/aim/channels">Weather</gml:name>
        <es:unknownEventsPossible>false</es:unknownEventsPossible>
        <es:publishedEvent xlink:href="#weatherRel1"/>
        <es:publishedEvent xlink:href="#weatherRel2"/>
      </es:EventChannel>
    </es:channel>
    <es:event>
      <es:EventMetadata gml:id="notam">
        <gml:name
codeSpace="http://www.opengis.net/ows7/aim/events">Notam</gml:name>
        <es:definition>A new NOTAM (NOtice To AirMen) is
available.</es:definition>
        <es:publishedOn xlink:href="#notamsRel1"/>
      </es:EventMetadata>
    </es:event>
    <es:event>
      <es:EventMetadata gml:id="weatherForecast">
        <gml:name
codeSpace="hhttp://www.opengis.net/ows7/aim/events">Forecast</gml:name>
        <es:definition>A new weather forecast is available.</es:definition>
        <es:publishedOn xlink:href="#weatherRel1"/>
      </es:EventMetadata>
    </es:event>
    <es:event>
      <es:EventMetadata gml:id="weatherObservation">
        <gml:name
codeSpace="http://www.opengis.net/ows7/aim/events">Observation</gml:name>
        <es:definition>A new weather observation is available.</es:definition>
        <es:publishedOn xlink:href="#weatherRel2"/>
      </es:EventMetadata>
    </es:event>
    <es:relationship>
      <es:EventChannelRelationship gml:id="notamsRel1">
        <es:eventEncoding
codeSpace="http://www.aixm.aero/schema/5.1/dnotam">Event</es:eventEncoding>
        <es:targetEvent xlink:href="#notam"/>
        <es:targetChannel xlink:href="#notamChannel"/>
      </es:EventChannelRelationship>
    </es:relationship>
    <es:relationship>
      <es:EventChannelRelationship gml:id="weatherRel1">
        <es:eventEncoding
codeSpace="http://www.eurocontrol.int/wx/1.1">Forecast</es:eventEncoding>
        <es:targetEvent xlink:href="#weatherForecast"/>
        <es:targetChannel xlink:href="#weatherChannel"/>
      </es:EventChannelRelationship>
    </es:relationship>
    <es:relationship>
      <es:EventChannelRelationship gml:id="weatherRel2">
        <es:eventEncoding
codeSpace="http://www.eurocontrol.int/wx/1.1">Observation</es:eventEncoding>
        <es:targetEvent xlink:href="#weatherObservation"/>
        <es:targetChannel xlink:href="#weatherChannel"/>
      </es:EventChannelRelationship>
    </es:relationship>
  </es:EventingInformation>
</wstop:TopicSet>
```

16.3 Event Metadata for SFE Service – XML Example

The following listing contains an example of a SFE service's event metadata contained as extension in a WS-Notification TopicSet. Such a TopicSet provides rudimentary information about the event channels supported by a service.

Listing 14: Example of Extended TopicSet with SFE Service Event Metadata

```
<wstop:TopicSet xmlns:wstop="http://docs.oasis-open.org/wsn/t-1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/wsn/t-1 http://docs.oasis-
open.org/wsn/t-1.xsd" xmlns:es="http://www.opengis.net/es/0.1"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:sfetns="http://www.opengis.net/ows7/sfe/channels">
  <sfetns:CameraPositions wstop:topic="true"/>
  <sfetns:DetectedChanges wstop:topic="true"/>
  <!-- the eventing information -->
  <es:EventingInformation gml:id="sfeEventingInformation"
xsi:schemaLocation="http://www.opengis.net/es/0.1 ../es.xsd">
    <es:channel>
      <es:EventChannel gml:id="positionChannel">
        <gml:name
codeSpace="http://www.opengis.net/ows7/sfe/channels">CameraPositions</gml:name>
        <es:unknownEventsPossible>false</es:unknownEventsPossible>
        <es:publishedEvent xlink:href="#positionRel1"/>
      </es:EventChannel>
    </es:channel>
    <es:channel>
      <es:EventChannel gml:id="changeChannel">
        <gml:name
codeSpace="http://www.opengis.net/ows7/sfe/channels">DetectedChanges</gml:name>
        <es:unknownEventsPossible>false</es:unknownEventsPossible>
        <es:publishedEvent xlink:href="#changeRel1"/>
      </es:EventChannel>
    </es:channel>
    <es:event>
      <es:EventMetadata gml:id="position">
        <gml:name
codeSpace="http://www.opengis.net/ows7/sfe/events">SensorPositionUpdate</gml:na
me>
        <es:definition>The position of a camera mounted on a moving
platform</es:definition>
        <es:publishedOn xlink:href="#positionRel1"/>
      </es:EventMetadata>
    </es:event>
    <es:event>
      <es:EventMetadata gml:id="change">
        <gml:name
codeSpace="http://www.opengis.net/ows7/sfe/events">VideoChangeDetected</gml:nam
e>
        <es:definition>A change was detected when comparing two video
streams.</es:definition>
        <es:publishedOn xlink:href="#changeRel1"/>
      </es:EventMetadata>
    </es:event>
    <es:relationship>
      <es:EventChannelRelationship gml:id="positionRel1">
        <es:eventEncoding
codeSpace="http://www.opengis.net/om/1.0">Observation</es:eventEncoding>
        <es:targetEvent xlink:href="#position"/>
        <es:targetChannel xlink:href="#positionChannel"/>
```



```
      </es:EventChannelRelationship>
    </es:relationship>
    <es:relationship>
      <es:EventChannelRelationship gml:id="changeRel1">
        <es:eventEncoding
codeSpace="urn:oasis:names:tc:emergency:cap:1.1">alert</es:eventEncoding>
        <es:targetEvent xlink:href="#change"/>
        <es:targetChannel xlink:href="#changeChannel"/>
      </es:EventChannelRelationship>
    </es:relationship>
  </es:EventingInformation>
</wstop:TopicSet>
```

17 Annex E – XML Schema for Event Model Revision

This Annex contains the XML Schema implementation of the Event Model version 0.3.

em.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:em="http://www.opengis.net/em/0.3"
targetNamespace="http://www.opengis.net/em/0.3" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <include schemaLocation="emEventSpecialization.xsd"/>
  <include schemaLocation="emEvent.xsd"/>
</schema>
```

emEvent.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:em="http://www.opengis.net/em/0.3"
targetNamespace="http://www.opengis.net/em/0.3" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <import namespace="http://www.opengis.net/gml/3.2"
schemaLocation="http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>
  <element name="DerivedEvent" type="em:DerivedEventType"
substitutionGroup="em:ComplexEvent">
    <annotation>
      <documentation>An event that is generated as a result of applying a
method or process to one or more other events.

When the procedure for combining base events is "disjunction", "conjunction",
"sequence", etc. then Luckham and Schulte (2008) call it a composite event. We
do not model this as a concrete specialization of DerivedEvent. Rather, we
treat it as a simple classification that is implied by the procedure used in a
DerivedEvent.</documentation>
    </annotation>
  </element>
  <complexType name="DerivedEventType">
    <complexContent>
      <extension base="em:ComplexEventType">
        <sequence>
          <element name="procedure" type="anyType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="DerivedEventPropertyType">
    <sequence minOccurs="0">
      <element ref="em:DerivedEvent"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
  <element name="Event" type="em:EventType"
substitutionGroup="gml:AbstractFeature"/>
  <complexType name="EventType">
```



```
    <complexContent>
      <extension base="gml:AbstractFeatureType">
        <sequence>
          <element name="eventTime">
            <complexType>
              <sequence>
                <element ref="gml:AbstractTimePrimitive"/>
              </sequence>
            </complexType>
          </element>
          <element name="spatialExtent" type="gml:GeometryPropertyType"
minOccurs="0">
            <annotation>
              <documentation>Geometry that best describes the spatial boundary
of the event.</documentation>
            </annotation>
          </element>
          <element name="attribute" minOccurs="0" maxOccurs="unbounded">
            <complexType>
              <sequence>
                <element ref="em:NamedValue"/>
              </sequence>
            </complexType>
          </element>
          <element name="relatedFeature" minOccurs="0" maxOccurs="unbounded">
            <complexType>
              <sequence>
                <element ref="em:EventFeatureRelationship"/>
              </sequence>
            </complexType>
          </element>
          <element name="relatedEvent" minOccurs="0" maxOccurs="unbounded">
            <complexType>
              <sequence>
                <element ref="em:EventEventRelationship"/>
              </sequence>
            </complexType>
          </element>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="EventPropertyType">
    <sequence minOccurs="0">
      <element ref="em:Event"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
  <element name="ComplexEvent" type="em:ComplexEventType"
substitutionGroup="em:Event">
    <annotation>
      <documentation>An event that is an abstraction of other events called its
members.</documentation>
    </annotation>
  </element>
  <complexType name="ComplexEventType">
    <complexContent>
      <extension base="em:EventType">
        <sequence>
          <element name="member" minOccurs="0" maxOccurs="unbounded">
            <complexType>
              <sequence>
                <element ref="em:EventEventRelationship"/>
```

```
              </sequence>
            </complexType>
          </element>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="ComplexEventPropertyType">
    <sequence minOccurs="0">
      <element ref="em:ComplexEvent"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
  <element name="EventFeatureRelationship"
type="em:EventFeatureRelationshipType" substitutionGroup="gml:AbstractObject"/>
  <complexType name="EventFeatureRelationshipType">
    <sequence>
      <element name="role" type="gml:CodeType" maxOccurs="unbounded"/>
      <element name="target" type="gml:FeaturePropertyType"/>
    </sequence>
  </complexType>
  <complexType name="EventFeatureRelationshipPropertyType">
    <sequence>
      <element ref="em:EventFeatureRelationship"/>
    </sequence>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
  <element name="NamedValue" type="em:NamedValueType"
substitutionGroup="gml:AbstractObject"/>
  <complexType name="NamedValueType">
    <sequence>
      <element name="name" type="gml:CodeType"/>
      <element name="value" type="anyType"/>
    </sequence>
  </complexType>
  <complexType name="NamedValuePropertyType">
    <sequence>
      <element ref="em:NamedValue"/>
    </sequence>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
  <element name="EventEventRelationship" type="em:EventEventRelationshipType"
substitutionGroup="gml:AbstractObject"/>
  <complexType name="EventEventRelationshipType">
    <sequence>
      <element name="role" type="gml:CodeType" maxOccurs="unbounded"/>
      <element name="target" type="anyType"/>
    </sequence>
  </complexType>
  <complexType name="EventEventRelationshipPropertyType">
    <sequence>
      <element ref="em:EventEventRelationship"/>
    </sequence>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
</schema>
```


emEventSpecialization.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:em="http://www.opengis.net/em/0.3"
targetNamespace="http://www.opengis.net/em/0.3" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <import namespace="http://www.opengis.net/gml/3.2"
schemaLocation="http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>
  <include schemaLocation="emEvent.xsd"/>
  <element name="ChangeEvent" type="em:ChangeEventType"
substitutionGroup="em:ComplexEvent"/>
  <complexType name="ChangeEventType">
    <complexContent>
      <extension base="em:ComplexEventType"/>
    </complexContent>
  </complexType>
  <complexType name="ChangeEventPropertyType">
    <sequence minOccurs="0">
      <element ref="em:ChangeEvent"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
  <element name="IndicationEvent" type="em:IndicationEventType"
substitutionGroup="em:Event"/>
  <complexType name="IndicationEventType">
    <complexContent>
      <extension base="em:EventType"/>
    </complexContent>
  </complexType>
  <complexType name="IndicationEventPropertyType">
    <sequence minOccurs="0">
      <element ref="em:IndicationEvent"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
  <element name="CancellationEvent" type="em:CancellationEventType"
substitutionGroup="em:ComplexEvent"/>
  <complexType name="CancellationEventType">
    <complexContent>
      <extension base="em:ComplexEventType"/>
    </complexContent>
  </complexType>
  <complexType name="CancellationEventPropertyType">
    <sequence minOccurs="0">
      <element ref="em:CancellationEvent"/>
    </sequence>
    <attributeGroup ref="gml:AssociationAttributeGroup"/>
    <attributeGroup ref="gml:OwnershipAttributeGroup"/>
  </complexType>
</schema>
```

Bibliography

[1] Hypertext Transfer Protocol -- HTTP/1.1, online at: http://www.w3.org/Protocols/rfc2616/rfc2616.html

[2] Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2, online at: http://docs.oasis-open.org/ws-rx/wsrm/v1.2/wsrm.html

[3] Luckham, D., Schulte, R. (2008), *Event Processing Glossary - Version 1.1.* Available at: http://www.ep-ts.com/component/option,com_docman/task,doc_download/gid,66/Itemid,84/ downloaded on February 13th, 2009.

[4] Web Services Security: SOAP Message Security 1.1 (WS-Security 2004) , OASIS http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[5] WS-Trust 1.4, OASIS http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.pdf

[6] WS-SecurityPolicy 1.3, OASIS http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/os/ws-securitypolicy-1.3-spec-os.pdf

[7] WS-Policy 1.2, OASIS http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/

[8] WS-SecureConversation 1.4, OASIS http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.pdf

[9] WS-Federation 1.2, OASIS http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.pdf

[10] SAML 2.0, OASIS http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip

[11] XACML 2.0, OASIS http://docs.oasis-open.org/xacml/2.0/XACML-2.0-OS-NORMATIVE.zip

