Open Geospatial Consortium

Date:2011-11-23

Reference number of this document: OGC 11-088r1

Category: Public Discussion Paper

Editors:Johannes Echterhoff, Thomas Everding

OGC®Event Service - Review and Current State

Warning

This document is not an OGC Standard. This document is an OGC Discussion Paper and is therefore 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, an OGC Discussion Paper should not be referenced as required or mandatory technology in procurements.

Document type:	OpenGIS®Discussion Paper
Document subtype:	NA
Document stage:	Approved for Public Release
Document language:	English

Contents

Page

Figures

Page

License Agreement

i. Preface

This Discussion Paper provides information on what has so far been called “Event Service” at OGC.

The presented work is supported by the European Commission through the ESS project (integrated project, contract number 217951) and the GENESIS project (integrated project, contract number 223996)[1].

ii. Document contributor contact points

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

Name	Organization
Johannes Echterhoff	International Geospatial Services Institute GmbH (iGSI)
Thomas Everding	Institute for Geoinformatics (IfGI), University of Münster, Germany

iii. Revision history

Date	Release	Editor	Primary clauses modified	Description
11-08-11	0.9	Johannes Echterhoff, Thomas Everding	all	initial draft for general review

iv. Future work

Additional feedback from Event Service implementers and users can be incorporated in the future, leading to a more detailed discussion of further aspects regarding the use of the Event Service.

[1]More information on the projects can be found at their project web pages: http://ess-project.eu/and http://www.genesis-fp7.eu/

Note that the Event Service is not an official standard. Its functionality is in the scope of the OWS PubSub SWG – though event processing functionality has already been agreed to not be in scope for OWS PubSub v1.0. Enablement of Event Processing is going to be addressed by a future OGC activity. This could be the OWS PubSub SWG itself but also a new working group.

Recommendations for future developments of the OWS PubSub SWG can be found in chapter 9.

Foreword

This document reflects the current use of WS-Notification (WS-N) in what has commonly been called *Event Service* in the OGC.

As explained in more detail in the introduction chapter, this document is a result of the work on the OGC SES Discussion Paper and the Event Architecture developed in OWS-7.

The work presented in this document does not cancel or replace other OGC documents.

This document may serve as input for the OWS PubSub SWG but it does not represent any normative result of that SWG.

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.

Introduction

This document reflects the current use of WS-Notification (WS-N) in what has commonly been called *Event Service*. It defines an interface to enable a publish-subscribe based communication between information producers and consumers but is not a standardized OGC service. This document describes its current status.

The Event Service was developed to satisfy the need for having relevant data available at an OWS pushed to a client as soon as it is available rather than having the client repeatedly poll the service. This need first came up in the OGC Sensor Web Enablement (SWE) domain working group (DWG). At that time, the Sensor Alert Service (SAS) interface was designed, which defined its own operations for supporting publish/subscribe based web service interactions rather than reusing existing IT standards. This design decision led to the development of the Sensor Event Service (SES), which is the successor of SAS and is based upon WS-Notification. However, it offers some additional functionality that was specifically designed to support requirements of the SWE domain. The service has subsequently been applied in the OGC Web Services Testbed Phases Six, Seven and Eight and also in the FAA SAA Dissemination Pilot. However, most of the SWE specific functionality was not used there. Rather, a pure WS-Notification based *Event Service* was deployed, which made use of the extension points defined by WS-Notification to support more sophisticated filter languages, to use additional event encodings and to define specific event channels. The Event Service supported various use cases and scenarios, from the SWE domain but also from the Aeronautical Information Management (AIM) domain, among others. The event encodings that were used in combination with the Event Service in these various applications are:

- O&M observations (v1.0)
- SAS alerts
- AIXM (v5.0 and v5.1)
- dNOTAM
- WXXM
- Event Model[2]

In 2010, a new Standards Working Group (SWG) was established at OGC: the OWS PubSub SWG. The purpose of this working group is to "*define an OGC Implementation Standard that enables publish/subscribe functionality for all OGC Web Services in a well-defined manner*" (OGC 10-182r4). The Event Service that was used in the mentioned OGC activities predates the activities of the PubSub SWG and is not available as a normative OGC standard. It is related to the PubSub SWG in that the SWG is going

[2]developed in OWS-6 (OGC 09-032) and improved and tested in OWS-7 (OGC 10-060r1) to encode complex events produced by the Event Service

to define support for publish/subscribe in OWS SOAP bindings and because the activities performed by the SWG indicate that WS-Notification is going to be used for enabling this support.

The figure below shows the timeline of the development from the Sensor Alert Service (SAS) to the results of the PubSub SWG.

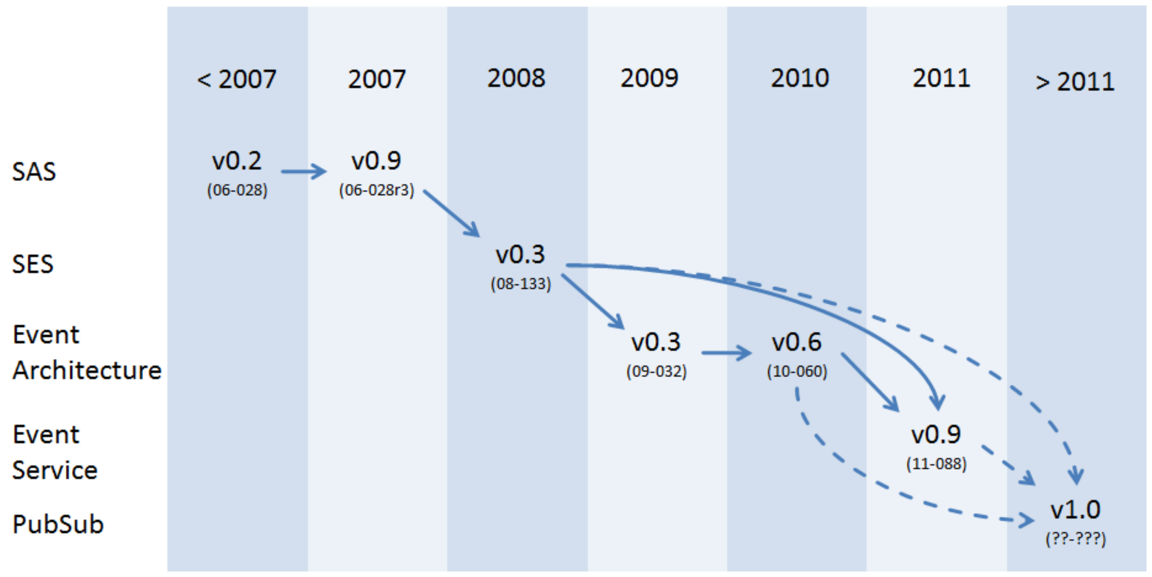

Figure 1 - Timeline of the development from the SAS to the results of the PubSub SWG

This document describes how the Event Service has been used so far in OGC standards based environments. In the terminology of the PubSub SWG it is a *Publisher* or *Broker*, depending on how the data that is published by the service is made available to it.

Note: the name *Event Service* is slightly misleading in the sense that data published by the service does not need to be pure event data, an event being defined as "*Anything that happens or is contemplated as happening at an instant or over an interval of time*" (OGC 09-032). Data can in fact be anything; it does not need to have a well-defined temporal aspect. The Event Service as described in this document has not been adopted by the PubSub SWG. The final standard produced by the PubSub SWG to enable publish/subscribe in OWS SOAP bindings may therefore not be fully compatible with the current use of the Event Service as summarized in this document.

OpenGIS®Event Service – Review and Current Status

1 Scope

This document can be used as reference of the usage of WS-Notification (WS-N) in what has commonly been called *Event Service* at OGC up until now.

This document does NOT represent a normative result of the OWS PubSub SWG. At the time of writing this document, the OWS PubSub SWG was still in the process of writing an OGC Standard to enable Publish Subscribe functionality in OWS.

2 References

The following documents contain provisions that, through reference in this text, constitute provisions of 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 document referred to applies.

OGC Event Pattern Markup Language (EML), OGC document 08-132

OGC FAA SAA Dissemination Pilot ER, OGC document 11-055

OGC Filter Encoding 2.0 Encoding Standard, OGC document 09-026r1

OGC Observations and Measurements - Part 1 - Observation schema, OGC document 07-022r1

OGC OWS-6 Secure Sensor Web Engineering Report, OGC document 08-176r1

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

OGC OWS-7 Event Architecture Engineering Report, OGC document 10-060r1

OGC Sensor Alert Service Implementation Specification, Best Practice Paper, OGC document 06-028r5

OGC Sensor Event Service Interface Specification, OGC document number 08-133

OGC SWE Service Model, OGC document 09-001

W3C Recommendation, Web Services Addressing 1.0 - Core, online at http://www.w3.org/TR/ws-addr-core

W3C Recommendation, Web Services Addressing 1.0 - SOAP Binding, online at http://www.w3.org/TR/ws-addr-soap

W3C Recommendation, XML Path Language (XPath) Version 1.0, online at http://www.w3.org/TR/xpath

W3C Recommendation, XML Path Language (XPath) 2.0 (Second Edition), online at http://www.w3.org/TR/xpath20/

3 Terms and definitions

Because the Event Service is based upon WS-Notification, this document uses WS-N terminology. Terms as used by the PubSub SWG are explicitly marked as such. If no qualifier (such as “PubSub” or “WS-N”)is provided for a given term, the WS-N terminology is assumed. This distinction is critical only to differentiate a PubSub Publisher from a NotificationProducer and a PubSub Sender from a WS-N Publisher.

For the purpose of this document, the definitions specified in section 2.1 of WS-BaseNotification, section 3 in WS-BrokeredNotification and section 2 in WS-Topics apply.

For disambiguation, the following table lists WS-Notification terms and their PubSub SWG synonym:

WS-Notification Term	PubSub SWG Synonym
Publisher	Sender
NotificationConsumer	Receiver
NotificationProducer	Publisher
NotificationBroker	Broker
Topic	Channel

4 Conventions - Abbreviated terms

AIM	Aeronautical Information Management
AIP	Architecture Implementation Pilot
AIXM	Aeronautical Information Exchange Model
CEP	Complex Event Processing
dNOTAM	digital NOTAM
DWG	Domain Working Group
EML	Event Pattern Markup Language
ER	Engineering Report
ESP	Event Stream Processing
EU	European Union
FAA	Federal Aviation Administration
FES	Filter Encoding Specification
GEOSS	Global Earth Observation System of Systems
HTTP	Hypertext Transfer Protocol
INSPIRE	Infrastructure for Spatial Information in the European Community
ISO	International Organization for Standardization
MEP	Message Exchange Pattern
NAT	Network Address Translation
NOTAM	Notice To Airmen
O&M	Observations and Measurements
OASIS	Organization for the Advancement of Structured Information Standards
OGC	Open Geospatial Consortium
OWS	OGC Web Services
PubSub	Publish - Subscribe

SAA	Special Activity Airspace
SAS	Sensor Alert Service
SES	Sensor Event Service
SFE	Sensor Fusion Enablement
SWE	Sensor Web Enablement
SWES	SWE Service Model
SWG	Standards Working Group
SWIM	System Wide Information Management
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WS-A	Web Services Addressing
WS-I	Web Services Interoperability
WS-N	Web Services Notification
WSDL	Web Services Description Language
WXXM	Weather Information Exchange Model
XML	Extensible Markup Language
XPath	XML Path Language

5 Event Service Functionality

This chapter describes the main functional aspects of the Event Service. The basic workflow is described first, followed by information on basic communication aspects. The PullPoint, a concept defined by WS-Notification (WS-N), is also explained because it helps circumventing problems with firewalls and information loss due to the client being offline. Finally, the important operations are described.

5.1 Workflow - Overview

The Event Service is a *NotificationBroker* following the WS-Notification standard. It therefore supports subscription creation like a basic *NotificationProducer*. However, in addition it implements the *NotificationConsumer* interface so that it can receive messages from publishers and thus acts as a standalone message/notification broker.

The basic workflow of a NotificationBroker is illustrated in the following sequence diagram.

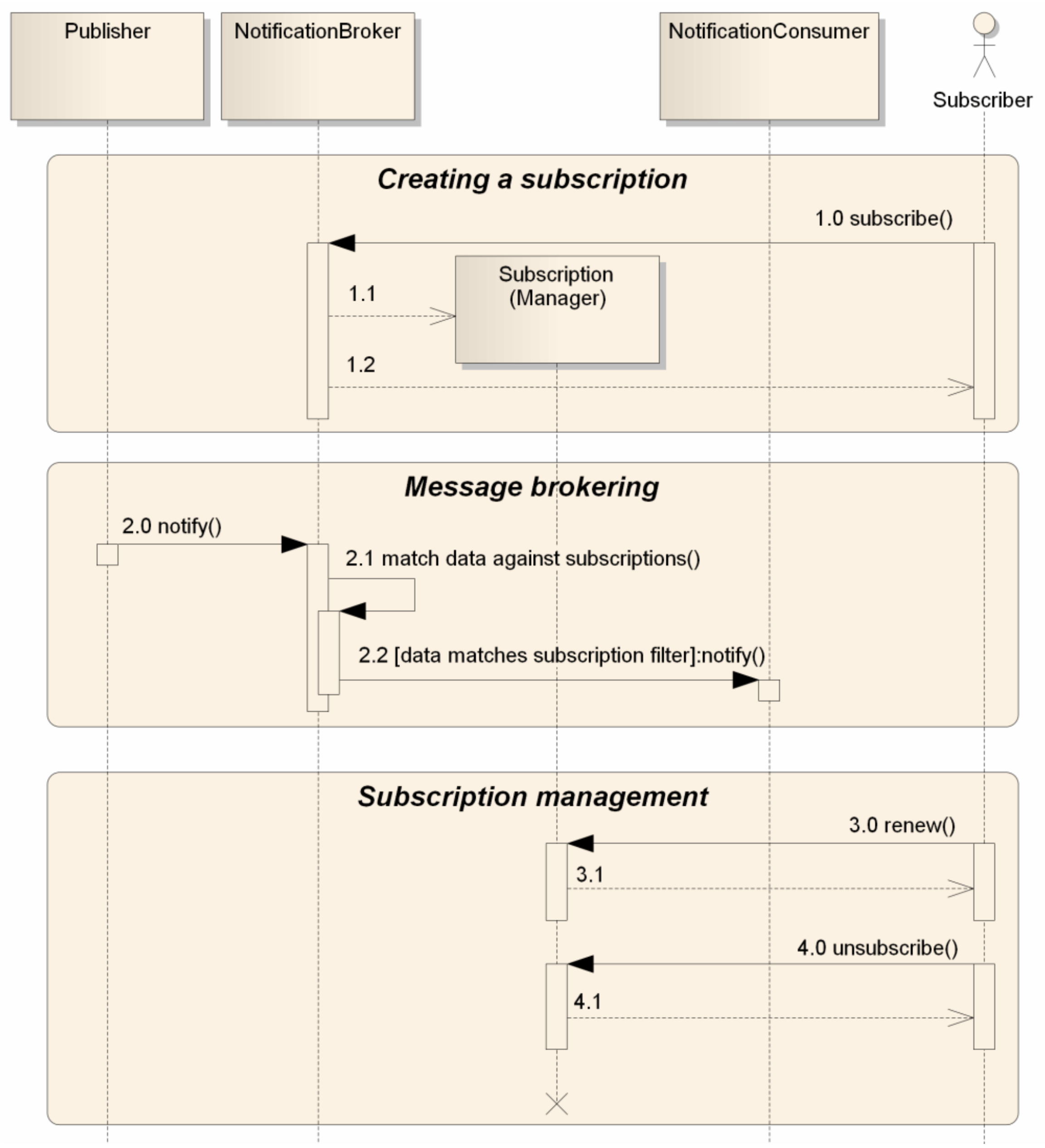

Figure 2: Basic workflow

The workflow consists of three parts: subscription creation, message brokering and subscription management.

To create a subscription, a *Subscriber* invokes the *Subscribe* operation (step 1.0). If the service accepts the subscription, it creates a logical web service entity called *SubscriptionManager* (step 1.1) and returns its address in the operation response (step 1.2). Management of subscriptions is explained later on.

A NotificationBroker / Event Service receives new data for distribution to interested clients via the Notify operation (step 2.0)[3]. Once a new message has been received this way, the data contained in the message is matched against the existing subscriptions (step 2.1). If the data matches the filter criteria of a given subscription, the broker sends it via a Notify message to the NotificationConsumer defined in the subscription (step 2.2).

Clients can manage their subscriptions via the operations defined in the WS-N BaseSubscriptionManager interface. The SubscriptionManager implements this interface and thus offers the Renew operation to extend the lifetime of the subscription it represents (step 3.0) and the Unsubscribe operation to terminate the subscription if desired (step 4.0).

Four operations - Notify, Subscribe, Renew and Unsubscribe - represent the core functionality of a NotificationBroker / Event Service[4].

5.2 Basic Communication

This section discusses aspects of basic communication that are relevant when using WS-Notification and related WS-* standards to enable Event Service functionality. SOAP as well as WS-Addressing (WS-A) are among the core standards in the general web service domain. Understanding how they work is a key factor for interoperability of Event Services.

WS-Notification uses WS-A to identify endpoints of system entities and to then invoke operations at these endpoints. The WS-A standard contains normative information on its use. In addition, numerous tutorials exist on the web. [OGC 09-032] explains essentials of WS-A in section 10.2. Detailed instructions on all aspects related to using WS-A is therefore not provided in this document. For that, the reader is referred to the mentioned documents.

One important aspect for the Event Service is that at the moment the only available binding for WS-A is its SOAP binding. Therefore, SOAP currently is a requirement for all WS-N based services and thus also the Event Service. The SOAP version used in Event Service implementations is 1.2, although both WS-Notification and WS-A support both SOAP versions. An Event Service can therefore also implement a SOAP 1.1 binding. For even more interoperability with clients, a service may support both SOAP 1.1 and 1.2.

[3]The workflow does not mention RegisterPublisher interactions because a NotificationBroker may allow Publishers to send new messages without having been registered beforehand - see section 9.2 for further details.

[4]The GetCurrentMessage operation is also not part of the workflow because even though the operation has to be implemented by a NotificationProducer, a NotificationProducer may choose to not support caching of the last message published on a given topic and thus simply return a fault upon a GetCurrentMessage request.

5.2.1 Address Information

A WS-A endpoint reference identifies the endpoint of a system entity. The endpoint reference requires a single URL that serves as the address of the system entity. In a simple setup, this information is sufficient to perform all necessary WS-N operations.

Similar to getting the URL of a service via a WSDL document, the URL provided by the address property of a WS-A endpoint reference can be used to send SOAP messages to it (usually via HTTP POST). If the receiving service does not require the use of WS-A - which can be governed via service policies, see section 5.2.3, then that message does not need to contain any WS-A specific header data.

However, a WS-A endpoint reference may also include additional parameters, called "reference parameters" - see [OGC 09-032] for further details. Event Service implementations have been using these parameters to manage multiple logically separate services - like SubscriptionManagers - at the same service URL. If such parameters are used in a WS-A endpoint reference, then SOAP messages sent to this endpoint need to include these parameters as per WS-A. It is then no longer possible to omit WS-A specific metadata in the message headers[5].

5.2.2 Communication Patterns

The following two sections discuss specifics of two basic message exchange patterns (MEP) that are required by WS-Notification: request-response based communication for the Subscribe operation and one way communication for the Notify operation.

5.2.2.1 Request Response Communication

Most web service operations, for example the WS-N Subscribe operation, are realized based upon an exchange of a message that requests information followed by a related message that returns that information, called response. Request and response message have a close relationship in that they both belong to the same communication. This MEP is usually performed synchronous but can also be asynchronous - see [OGC 09-032] for further details.

SOAP via HTTP supports synchronous request-response. This is basic and well understood web service functionality. Together with WS-A, also asynchronous request-response is possible. WS-A defines SOAP headers which can be included in a SOAP request message to instruct the receiver of that message where to send the response/fault to. WS-A defines two special URIs that enforce synchronous communication (the *anonymous-URI*) and dropping of a response/fault message (the *none-URI*) - see [OGC 09-032] for further details.

---

[5]Because the parameters need to be included as SOAP header elements with the *wsa:IsReferenceParameter* attribute being set to true

Service policies can be used to indicate if WS-A is required in communications with a service and if the WS-A anonymous-URI must or must not be used - see section 5.2.3for further details.

5.2.2.2 One Way Communication

Performing a request-response based operation - like a WS-N Subscribe - with SOAP via HTTP without using WS-A is relatively simple, as the semantics of the MEP[6] map nicely. However, sending a one-way message without expecting a response like it is done in the WS-N Notify operation requires specific consideration. The SOAP HTTP bindings only define request response based communication (see SOAP 1.1 and SOAP 1.2 Adjuncts). The handling of one-way messages therefore needs special consideration. The WSDL 1.1 SOAP (1.1) binding appears to just ignore that the SOAP 1.1 HTTP binding does not explicitly support one-way messages[7]. The WSDL 2.0 SOAP 1.1 binding is a bit more specific in that it says that in the case of an in-only MEP via SOAP 1.1 "*the HTTP response is undefined*". It points the reader to the WS-I Basic Profiles or other specifications that may provide the missing information. For SOAP 1.2, WSDL 2.0 defines how it works for in-out, in-only and robust-in-only (with the option to deliver a fault if an exception occurred) and therefore covers the one way case. The standard explicitly mentions that other standards like WS-A may override the default behavior.

The WS-N Notify operation itself does not define any fault that is specific for that operation. However, other faults - for example SOAP or WS-A specific - may occur. Furthermore, additional faults may be defined by standards that extend the WS-N functionality. A WS-N Publisher should therefore handle the WS-N Notify operation according to the robust-in-only MEP - even if SOAP 1.1 is used.

The following list provides instructions if and how to return a fault in a one way communication based upon the type of fault and the use of WS-A. The instructions are listed in descending priority.

- "*If a SOAP envelope does not contain any WS-Addressing header blocks, or contains WS-Addressing header blocks that do not include any soap1x:mustUnderstand="1" attributes, and the RECEIVER chooses to ignore them, then any response (normal or fault) SHOULD be transmitted. If it is transmitted then it is transmitted on the HTTP Response message (if available).*" (see requirement 1145 in both WSI-Basic Profile 1.2 and 2.0)

[6]Message exchange patterns define how communication between system entities is performed. The simplest one is to just send a message without expecting any form of response, including a fault. This is called *one-way* message by WSDL 1.1 and *in-only* by WSDL 2.0. The case that a fault is communicated for a one-way message is not covered by WSDL 1.1 but WSDL 2.0. There, the according MEP is called *robust-in-only*. The common case where a request message is sent and a response is expected is called *request-response* in WSDL 1.1 and *in-out* in WSDL 2.0.

[7]A W3C Note exists which describes an extension for the SOAP 1.1 HTTP binding to support a *request-optional-response* MEP: http://www.w3.org/TR/soap11-ror-httpbinding/. This could support one-way messages but because it is not normative it is not integrated in other relevant standards (like WSDL or the Basic Profiles).

So if WS-A is not required by a NotificationConsumer[8] and the WS-N Publisher chooses to not use it at all, then a one-way message is sent to a NotificationConsumer via an HTTP request and any possible fault will be returned via the HTTP response.

- If a SOAP envelope does contain WS-Addressing header blocks that the receiver understands, then in case that:
  - the WS-A [reply endpoint] is set to http://www.w3.org/2005/08/addressing/noneand the WS-A [fault endpoint] is not set, then the fault message is discarded
  - the WS-A [fault endpoint] is set to http://www.w3.org/2005/08/addressing/nonethen the fault message is discarded, regardless of the WS-A [reply endpoint] setting
  - the WS-A [reply endpoint] is set to http://www.w3.org/2005/08/addressing/anonymousand the WS-A [fault endpoint] is not set then send the fault message in the HTTP response
  - the WS-A [fault endpoint] is set to a valid service URL (and neither the WS-A none-URI nor the WS-A anonymous-URI) then send the fault message to that URL, regardless of the WS-A [reply endpoint] setting

Note that in case that WS-A is enforced (either by the NotificationConsumer through policy statements or the WS-N Publisher through WS-A headers with mustUnderstand=true on all of them) the Notify request shall contain the WS-A headers as defined by WS-A Metadata for the robust-in-only MEP. This means that the destination, action, reply and/or fault endpoint as well as the message id have to be present in the Notify message.

5.2.3 Policies

Service descriptions - for example WSDL documents - can indicate behavior that is supported, not supported, or required via policy statements included in these descriptions.

The behavior regarding WS-A can be described via WS-A policies - defined in WS-A Metadata. The policies indicate if WS-A is supported by the service and also if its use is required. Note that the WS-I Basic Profiles (see section 3.7.16 in the according documents) specify how this information can also be defined on the level of individual operations. WS-A policies can further indicate if response addresses must or must not have the WS-A anonymous-URI as value. See [OGC 09-032] for further details on WS-A policies.

[8]This can be indicated via WSDL metadata provided in the consumer endpoint reference of a subscription, see WS-Addressing Metadata for further details on including WSDL metadata in EPRs. If the endpoint reference does not contain such metadata then the WS-N Publisher does not have exact knowledge if WS-A is not supported, supported or required. The Publisher may just try to send a Notify message without WS-A information and - in case that a fault indicates that WS-A headers are required - can simply include the information and resend the message. This requires that a Publisher supports more WS-A functionality.

5.3 PullPoint

A WS-N NotificationProducer per default invokes the Notify operation on WS-N NotificationConsumers to deliver data matching the filter criteria of the respective subscriptions. As outlined in section 8.1, this push-based communication may be problematic for simple clients that are not web services. In cases where the client cannot be reached from another service (for various reasons such as firewall and being offline) a workaround is needed. WS-Notification defines the concept of a PullPoint, which is a web service that acts as a proxy for a client.

A PullPoint is a NotificationConsumer and thus can be used in subscriptions as the endpoint to which Notify messages are sent. A NotificationProducer can deliver data matching a subscription to the PullPoint and clients can retrieve that data whenever they want to.

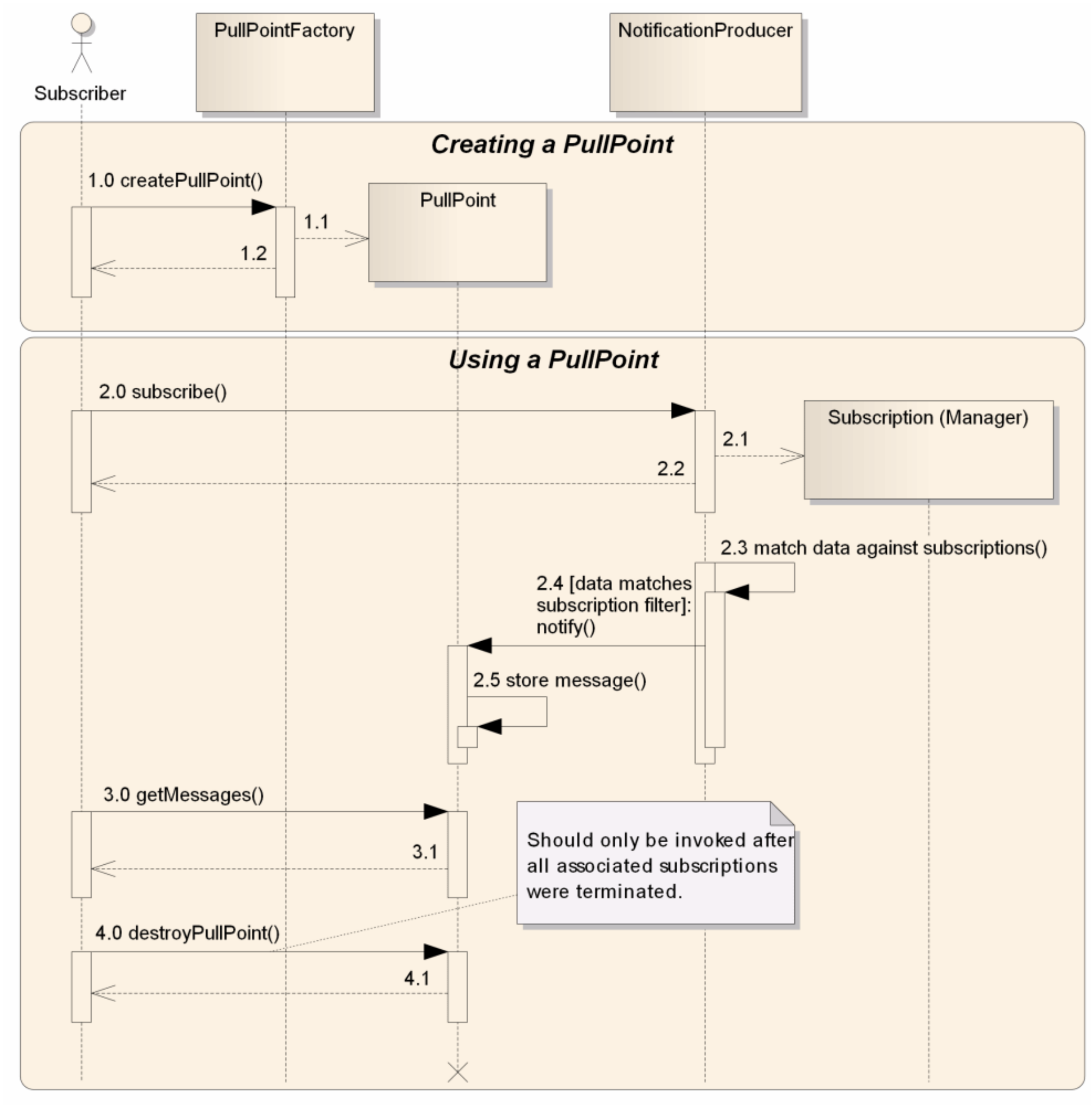

Figure 3: Creating and using a PullPoint

PullPoints are created at web services that act as PullPointFactories (steps 1.0, 1.1 and 1.2). This factory may but does not need to be the same service entity as the NotificationProducer. The response of the CreatePullPoint operation contains the WS-A endpoint reference of the new PullPoint. The client can use this endpoint reference in a subscribe request (step 2.0, 2.1 and 2.2). Whenever data matches the subscriptions filter criteria the NotificationProducer sends it to the PullPoint via a Notify request (steps 2.3 and 2.4)[9]. The PullPoint then stores the message (step 2.5). Clients can retrieve stored

[9]WS-Notification allows a PullPoint to ignore messages received via the WS-N Notify operation if it uses some implementation specific way to receive messages. However, it is strongly recommended that a PullPoint always stores and returns the messages that it received via the Notify operation.

messages from a PullPoint via the GetMessages operation (steps 3.0 and 3.1). Retrieved messages are removed from the PullPoint, i.e. a subsequent GetMessages request will not return a message that has already been returned. If the client no longer needs the PullPoint then it can remove it via the DestroyPullPoint operation (step 4.0 and 4.1). This operation should not be invoked before all subscriptions that use the PullPoint as consumer endpoint reference have been terminated.

Note that a single PullPoint can be used in multiple subscriptions. Messages that belong to different subscriptions can be differentiated via the SubscriptionReference metadata that NotificationProducers may include in NotificationMessages - see WS-BaseNotification section 3.1 for further information on which metadata elements can be included in messages. Unfortunately, WS-Notification does not define a way for subscribers to ensure that NotificationMessages contain any of these elements. This can be defined by OGC but has not been done so far.

5.4 Operations

The following sections contain information on the core WS-N operations that are used in the workflow illustrated in Figure 2. Basic implementation details for these operations are provided in the respective WS-N standards and not repeated here. The following sections rather discuss the respective operations in general.

5.4.1 Subscribe

The Subscribe operation implements the request-response MEP.

A Subscriber invokes the operation by sending a Subscribe request message to a NotificationProducer. The request has to contain a WS-A endpoint reference of the system entity to which data matching the subscription shall be sent via Notify messages (see section 5.4.2).

The request may contain filter statements. If no such statement is provided then all data made available by the NotificationProducer is of interest to the subscription. Otherwise, the data needs to be matched by the service against the filter criteria. So far, only message content filters and topics were applied in Event Service use cases. See chapter 6and WS-Notification for further information on filtering performed by an Event Service.

The Subscriber may also provide an initial termination time. This has been used only with absolute date and time values, not with duration relative to server time. Further handling of the initial termination time is defined by WS-Notification.

Furthermore, subscription policies may be provided but these have also not been used so far.

The response contains the WS-A endpoint reference of the SubscriptionManager that handles the new subscription. Renew and Unsubscribe requests are sent to this endpoint. The response may contain the current time of the server. However, this response property has not been used so far. The termination time of the subscription, on the other hand, can also be included in the response and this property has been used in implementations.

If the request does not specify an initial termination time then either the service sets it in the response or the subscription lives until it is explicitly terminated. It is therefore recommended to set a desired termination time in the request. If the service is unwilling or unable to honor the requested termination time, then an UnacceptableTerminationTimeFault will be returned, which should contain a hint that indicates acceptable values for the termination time. WS-Notification unfortunately does not define how a service can indicate in its service description/metadata which termination time values are in general acceptable for subscriptions (the OWS PubSub SWG should consider specifying support for this). Clients may use the Renew operation to extend the lifetime of a subscription before it terminates.

5.4.2 Notify

Data is delivered to and from a NotificationBroker via the WS-N Notify operation. The actual data is contained inside a NotificationMessage element.

A NotificationMessage may contain metadata about the actual data, such as which subscription it matched, on which topic it was published and which NotificationProducer sent the message. The metadata is therefore applicable in messages sent from a NotificationProducer to a NotificationConsumer. It apparently is not designed for the case of a simple Publisher sending a message to a NotificationBroker.

Multiple pieces of matching data can be transported in one Notify message as it can contain one or more NotificationMessage elements. For a NotificationConsumer, this is the same as receiving multiple Notify messages with just one NotificationMessage element each - at least with respect to how individual NotificationMessages are handled by the consumer.

The Notify operation does not define a response. It implements the one-way MEP. However, an exception may occur while the Notify operation is invoked. How such fault messages can be communicated is discussed in section 5.2.2.2.

5.4.3 Renew

The Renew operation allows clients to reset the termination time of a subscription. A Renew request is always sent to the endpoint of the SubscriptionManager - the one that was returned in the Subscribe operation response. The request contains the suggested termination time. It has been used only with absolute date and time values, not with duration relative to server time. See the documentation of the Renew operation in WS-Notification for further information on handling a Renew request. The operation response repeats the new termination time, if it has been reset as requested.

Note that WS-Notification does not define how a service can indicate in its service description/metadata the maximum duration that clients may extend the lifetime of their subscription with (again, this should be considered by the OWS PubSub SWG).

5.4.4 Unsubscribe

The WS-N Unsubscribe operation is a simple means to terminate a subscription.

A Subscriber needs to ensure that it stores the endpoint reference of a subscription (rather: its SubscriptionManger) if the Subscriber intends to terminate or renew it after it was created. WS-Notification does not define means to retrieve subscription metadata unless the endpoint of the according SubscriptionManager is known. Clients thus cannot get information about the endpoints of SubscriptionManagers that were created for them or for others - at least not with any means defined by WS-Notification.

6 Filtering

The Event Service offers two possibilities to restrict the set of events that are sent to a receiver. One is the concept of channels (also known as topics) which is described in the next section. The approach discussed here is filtering which allows forwarding or dismissing an event based on its content.

The filter that shall be applied is an optional component of the subscribe request (see section 5.4.1). WS-Notification does not restrict the way the filter is encoded; however, the Sensor Event Service (SES) specification defines three filter levels that are also applicable for the Event Service:

1. XPath for filters based on the event structure and simple checks of property values.
2. OGC Filter Encoding for more sophisticated property checks including spatial and temporal operators.
3. OGC Event Pattern Markup Language for Complex Event Processing (CEP) and Event Stream Processing (ESP) capabilities.

Which language is used has to be specified in the dialect attribute of the filter definition. Currently, there is no strict definition of the correct value for each filter language. Until now the Event Service used the dialect definitions as defined in the SES specification which are:

- “http://www.w3.org/TR/1999/REC-xpath-199911” for XPath,
- “http://www.opengis.net/ses/filter/level2” for the OGC Filter Encoding and
- “http://www.opengis.net/ses/filter/level3” for the EML.

Especially the last two dialect definitions are not optimal. It is recommended to use the XML namespace (if available) or any other URL that clearly identifies the language. For the OGC Filter Encoding version 2.0 this would be “http://www.opengis.net/fes/2.0”.

This way also the version of the filter language can be specified and changed without the need to change the Event Service / PubSub specification.

6.1 XPath

The XML Path Language (XPath) is developed by the W3C. It is currently available in version 2.0 (December 2010). However, the implementation in the Event Service used the older version 1.0. In general, it can be used to address specific parts in an XML document. An XPath based filter is evaluated to true (which causes the XML document to be forwarded to the subscription's NotificationConsumer) if the defined path can be found in the XML representation of an event.

It can for instance be used to filter for specific identifiers. The following XPath expression checks if the identifier of an airspace has a specific value and a specific code space:

```
//aixm:Airspace[identifier='AirspaceID']/gml:identifier[@co
deSpace='code.space.uri']
```

Only events containing the following aispace definition will pass this filter statement:

```
<aixm:Airspace gml:id="airspace_01">

   <gml:identifier
codeSpace="code.space.uri">AirspaceID</gml:identifier>

   …

</aixm:Airspace>
```

For a correct and uniform resolution of XPath expressions some restrictions on the expression context have to be made. In detail, any XML that is used as event container shall be excluded and each XML document describing an event shall describe exactly one event and shall have no more than one root element. A more formal description can be found in the OGC SWE Service Model (OGC 09-001) in section 17.2.3.1.

6.2 OGC Filter Encoding

In most use cases the OGC filter encoding was used. Even though the SES specification uses an extended version of FES 1.0, the recent version 2.0 (OGC 09-026r1) was implemented. This was done because it was an official release that contained the additional capabilities of the extension made in the SES and was also used as filter language for Web Feature Service requests in many use cases.

The Filter Encoding allows to define filter using comparison operators (e.g. greater than, between, less than, …), spatial operators (e.g. intersects, within distance, contains, …) and temporal operators (e.g. after, during, before, …). Complex filters can be defined by combining multiple statements using the logical operators AND or OR. For instance in the aviation use cases a common filter was: "*Give me all events that occur during my flight time and thatare located within a buffer of 100 nautical miles around my flight path.*"

For the addressing of event properties the filter encoding recommends to use XPath if the event is encoded in XML. In the example above this would mean that the geometry position in the event would be referenced by an XPath expression like "`//aixm:Airspace/gml:boundedBy`". As an alternative one can define keywords to refer to specific event properties like "geometry". This can be simpler to implement, especially when complex properties like the geometry need to be accessed but the XPath version is more flexible. In addition, the mapping of keywords to their XPath equivalent has to be document appropriately to avoid confusion. Note that the restrictions regarding the XPath context described above also apply here.

6.3 OGC Event Pattern Markup Language

The Event Patter Markup Language (EML) is available as an OGC Discussion Paper (OGC 08-132). It can be used to define rules for Complex Event Processing (CEP) and Event Stream Processing (ESP). It enables theinclusion of multiple events into the filtering process and even to derive new information.

EML was used, for example, in the OWS-7 SFE thread to detect when a vehicle entered a specific area. Instead of performing a rather simple and inexact spatial filter that just detects location events to be within the area, it was checked if an event on the outside was followed by one on the inside. This way a vehicle entry event was derived and forwarded instead of sending each position update from the inside.

Especially in situations when a forwarded event is used as a trigger for further processing by the receiver it can be important to reduce the event output to the necessary minimum.

Further discussions about the filtering of events can be found in the SES specification (OGC 08-133), the OWS-7 Event Architecture ER (OGC 10-060r1), and the OGC SWE Service Model (OGC 09-001).

6.4 Topics

Data at an Event Service is usually published and made available to subscription matching processes as one big cloud of data. New data is usually processed as soon as it is available to the service. Topics as defined by WS-Topics provide a way to structure the data cloud. In a sense, they provide a specific view upon the data cloud. Views from the database domain are a similar concept. In addition, TV channels also represent the concept. There, a Sports channel will publish Sports related news, while the news published on a channel like CNN will have a more general scope.

In any case, this subsetting of the data cloud can help improve performance and ease the task of creating subscriptions. This is explained in more detail in section 6.3 of [OGC

10-060r1]. An example of a subscription that targets a specific topic is provided in section 10.4.4 of [OGC 09-032] (listing 10).

As we can see, an Event Service may support both content and topic based filtering. If a subscription contains filter expressions of both type, it is up to the Event Service implementation which filter is applied first. Optimizations can be performed – but that is entirely up to the service. Data only matches a subscription if all filter criteria are satisfied. So in case that a subscription is listening to topic X and expects data to have content Y, then new data has to satisfy both criteria in order to be sent to the subscription's NotificationConsumer.

The modeling of topics according to WS-Topics is described in section 10.4.2 of [OGC 09-032]. Domains may design static topics in so called topic namespaces. The semantics of a given topic are defined and documented by the domain, such as which data is published on the topic when and in which format and encoding. The SWE Service Model standard, for example, defines a topic namespace. In the SAA Pilot (see OGC 11-055), topics were defined on which specific heartbeat and airspace schedule events were published.

The topics supported by an Event Service instance can be indicated in the so called TopicSet. Again, please refer to [OGC 09-032] for further details. However, accessing the TopicSet with WS-Notification based mechanisms is only possible if a NotificationProducer supports resource properties – see the WS-BaseNotification standard section 4.1 for further details. The OWS PubSub SWG should therefore consider improving the provision of topic metadata by a PubSub service.

7 Use of Event Services

This chapter provides an overview of projects and experiments that made use of the Event Service at the time of writing. It includes the recent and ongoing OGC testbeds and some EU funded projects. This list reflects the knowledge of the authors and may not be exhaustive.

7.1 OGC Aviation Activities

At the OGC Web Services Testbed Phase 6 (OWS-6) the Aviation thread was introduced. Its general aim is to test and demonstrate the use of OGC standards to solve aeronautical use cases. As a part of this activity, the Event Service was introduced to provide a publish-subscribe based access to aeronautical events encoded in AIXM. This work continued in OWS-7: more sophisticated filtering capabilities were implemented and management functionality for subscriptions (e.g. unsubscribe) was tested. Furthermore, the support for weather events encoded in WXXM was added.

In the ongoing OWS-8 Aviation thread the main topics regarding the Event Service are the enhancement of the filtering functionality and of the delivery methods. The EML (see chapter 6) is used to build so called dynamic spatial filters that are automatically updated based on the position of an aircraft. PullPoints (see section 5.3) are implemented to circumvent delivery problems that were noticed during the previous testbeds.

In parallel to the OWS Aviation threads also the FAA SAA Dissemination Pilot was performed. Its organization was similar to the OWS testbeds but it focused on a single topic: the integration of the SWIM services developed by the FAA with OGC web services. Especially the dissemination of Special Activity Airspace (SAA) information was of interest. The Event Service was used for the PubSub based dissemination of this information. The according work tasks encompassed the connection to the SWIM data sources, the enrichment of the event messages for the filtering process and their encoding. Further related topics were the definition and implementation of a mechanism to test whether the Event Service is still running (see section 8.2) and the provision of event channels (see section 6.4).

7.2 Further OWS Threads

Besides the Aviation threads, the Event Service was topic of further activities within the OWS testbeds. The most important ones are the Eventing cross threads of OWS-6 and OWS-7 that discussed the problem of publish-subscribe from a thread independent perspective. In these threads two reports were produced that served as input for the OWS PubSub SWG. The reports are available at the OGC with the document numbers 09-032 and 10-060r1.

In the Sensor Fusion Enablement (SFE) thread of OWS-7 the Event Service was used to trigger a complex video comparison process. A filter encoded in EML was used to detect when a vehicle with a mounted camera entered a predefined area where historic video data was available for comparison. The Event Service output was newly derived information encoded in a format defined and improved in the Eventing cross threads of OWS-6 and OWS-7.

7.3 OSIRIS

The OSIRIS project was a three year EU funded project starting in 2006. Its aim was to develop a web service based architecture for risk monitoring and crisis management. It included work items on the improvement of the OGC publish-subscribe services and largely contributed to the development of the Sensor Event Service (SES, OGC 08-133) and Event Pattern Markup Language (EML, OGC 08-132) specifications.

7.4 GENESIS

GENESIS is an EU funded research project that started in 2008 and went over a period of three years. The main objective was to develop a generic integrated platform for environmental management and decision making. This platform was developed in synergy with European and global harmonization initiatives like INSPIRE and GEOSS and is based on international standards such as those from the OGC, W3C, ISO and OASIS.

In the new technology assessment work packages also experiments integrating an Event Service were performed. The focus was on the architectural integration with other web services and the implementation of an event driven architecture as well as on the exploitation of event processing techniques. While the Event Service was not integrated in the final GENESIS platform, the experimentation results were used as an input to other activities like the GEOSS AIP-3 Pilot (biodiversity) and the recent OWS testbeds.

7.5 ESS

ESS is an EU funded research project which aims to integrate data from various sources into a common information management and communication platform, develop portable and mobile smart communication elements for supporting the management and coordination of emergency operations and integrate ad hoc networking technology of intelligent sensors for addressing emergency and crisis management requirements.

Event Service functionality is used to distribute relevant information - e.g. sensor measurements, processing results and system messages - to the appropriate users and system entities as soon as the information is available.

8 Common known problems

In this section some problems are described that were faced periodically across the different use cases and pilots involving an Event Service.

8.1 Push Delivery Problems

One common problem when integrating the Event Service with other components was to establish a connection for the push based communication (see Notify operation in section 5.4.2). In general, we can say that aNotificationConsumeracts as a web service and the NotificationProducer or Publisher acts as a client that establishes a connection to the NotificationConsumer to deliver a message. Therefore the NotificationConsumer has to be accessible from the outside. In this context problems arise from addressing and security constraints.

When the NotificationConsumer is connected to the Internet via an NAT (Network Address Translation) device like a router, it cannot be addressed directly from the outside. Its external address is the one of the NAT device which has to be configured to forward incoming requests to a specific address in the local network. Further complications for the connection are introduced by firewalls that are used to secure many networks. A more detailed discussion on this problem can be found in the OWS-6 Secure Sensor Web Engineering Report (OGC 08-176r1), chapter 12. A solution to this issue is the use of PullPoints as described in section 5.3.

8.2 Detecting Service Failure

Another problem for NotificationConsumers is to detect if an Event Service is operational. Usually, the only time when the NotificationConsumer is contacted by the

Event Service is when a new notification matches the according subscription criteria. But as long as no notifications are sent, the NotificationConsumer cannot determine whether simply no notifications matched the subscription or if the service is down. Especially in environments where a reliable communication is essential, a service failure needs to be detected as soon as possible.

In the OGC FAA SAA Pilot study this problem was discussed and an experimental solution implemented. The discussed solutions were:

- a periodic request of the server state via GetCapabilities or using a dedicated new operation
- a periodic heartbeat event sent by the service that clients (NotificationConsumers) can (be) subscribe(d) to; the heartbeat rate is governed by the service
- specific heartbeat events whose rate is governed by the clients

Further details on this issue are documented in the OGC FAA SAA Dissemination Pilot Engineering Report (OGC 11-055).

8.3 Use of Time Zones

Another problem that appears from time to time is the interpretation of time values. Especially when times for the automatic termination of subscriptions and other service resources are used, a unified interpretation of time values is critical. Whereas the encoding of pure time stamps was not a problem in past experimentation activities, the use of time zones caused some issues.

In general, it is recommended to always add the time zone information to all time values in the inter-component communication and to make sure that all components are capable of processing this information.

9 Recommendations

The following recommendations are intended to facilitate further developments related to PubSub functionality in OGC, especially as input to the OWS PubSub SWG. The first five recommendations relate to publish-subscribe communication where the last two are of general interest.

9.1 Handling of One Way Communication

As described in section 5.2.2.2, services that receive notification messages may detect and be able to communicate an exception related to the receipt of this message. Therefore, instead of a simple *in-only* MEP (see section 5.2.2.2), the Notify operation should rather be used according to the *robust in-only* MEP. A NotificationProducer may even be able to indicate which MEP it supports via its service description/metadata.

Furthermore, even if no exception occurred, HTTP based communication is inherently request-response based. Thus, even though an actual response document does not need to be sent back to the entity that invoked the Notify operation, still setting the HTTP response code correctly and handling the HTTP connection correctly should be sufficiently defined. The WS-I BasicProfile should be leveraged as much as possible there.

9.2 RegisterPublisher Operation

The RegisterPublisher operation is used to register a new Publisher at a NotificationBroker. According to WS-BrokeredNotification, a NotificationBroker has to implement this operation. However, a NotificationBroker may choose to allow Publishers to send Notify messages to the service without having been registered beforehand. Thus, in the simplest case the RegisterPublisher operation does not need to be used.

The RegisterPublisher operation is required to support demand-based publication as defined by WS-Notification. However, this is an advanced concept that has not been used so far in Event Service implementations.

The RegisterPublisher operation can be used to ensure that a given Publisher interacts with the NotificationBroker as agreed in the registration. However, the necessary details to establish this level of security are not defined by WS-Notification.

The operation can also be used to provide metadata on the event source to the NotificationBroker and to communicate the format and encoding in which the Publisher will send event messages to the NotificationBroker. This can be useful to establish a more detailed contract between Publisher and NotificationBroker.

9.3 Providing Topic Metadata

The topics supported by an Event Service instance can be indicated in the so called TopicSet. However, accessing the TopicSet with WS-Notification based mechanisms is only possible if a NotificationProducer supports resource properties – see the WS-BaseNotification standard section 4.1 for further details. The OWS PubSub SWG should therefore consider improving the provision of topic metadata by a PubSub service.

9.4 Provision of WSDL Description in Endpoint References

In order to facilitate binding of clients to SubscriptionManagers and of NotificationProducers as well as Publishers to NotificationConsumers, the respective endpoint references should include WSDL metadata for the respective endpoint. This would facilitate detection of the supported SOAP version(s) and the service policies regarding the use of WS-Addressing (is it required, supported or not supported and are special URIs allowed).

9.5 Filter Language Identifiers

The identifier URLs for common filter languages (e.g. FES 2.0) to be used in (WS-N based) subscriptions should be clearly defined.

9.6 Time Zones

The handling of time zones should be defined, especially if times without time zone are allowed and how these should be interpreted. Usually, this means that server local time is used but if server and client are in different time zones, this can cause confusion.

9.7 XPath Usage

XPath may be used as standalone filter language. However, an abbreviated version is also used by the OGC Filter Encoding Specification. It is essential to clearly define the context of XPath based filter expressions. This also applies to value references used in FES. Furthermore, it is essential to clearly define how namespace prefixes for elements in XPath expression are handled, especially where the namespace binding has to be included (e.g. together with the XPath expression or in the context of the element that contains that expression).