Open Geospatial Consortium, Inc.

Date: 2010-06-30

Reference number of this document: OGC 10-061r1

Category: Public Engineering Report

Editors: Johannes Echterhoff, Ingo Simonis

OWS-7 Dynamic Sensor Notification Engineering Report

Warning

This document is not an OGC Standard. This document presents a discussion of technology issues considered in an initiative of the OGC Interoperability Program. This document does not represent an official position of the OGC. It is subject to change without notice and may not be referred to as an OGC Standard. However, the discussions in this document could very well lead to the definition of an OGC Standard. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

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

Preface

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 Dynamic Sensor Notification Engineering Report

1 Introduction

1.1 Scope

This OGC™ document is applicable to scenarios where moving sensors need to be tracked and their entry into an area of interest needs to be detected.

The document presents a detailed discussion of different approaches for encoding tracked object position.

Two approaches for implementing dynamic sensor tracking and notification are described, one based on the Sensor Alert Service specification and the other based on the Sensor Event Service specification.

An overview of standards and specifications relevant for and related to dynamic sensor tracking and notification is provided.

1.2 Document contributor contact points

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

Name	Organization
Ingo Simonis (editor)	International Geospatial Services Institute GmbH (iGSI)
Johannes Echterhoff (editor)	International Geospatial Services Institute GmbH (iGSI)
Angela Amirault	Compusult Limited
Paula Luscombe	Compusult Limited
Thomas Everding	University of Muenster - Institute for Geoinformatics

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2010-03-05	0.1.0	IS, JE	all	initial draft
2010-05-27	1.0.0	JE	all	revised all sections
2010-06-08	1.0.0	JE, IS	all	finalized report

1.4 Future work

During the testbed, the following work items were identified that could be addressed in the future:

- **Encoding of Position for Tracking** – Two approaches for encoding position data used for tracking were implemented in the testbed. This report discusses the advantages and disadvantages of these and additional encoding approaches in detail. Further work on dynamic sensor tracking should take into account the results of the discussion. For example, a new application schema could be developed to define models used for tracking. However, as tracking of moving objects or rather the identification of the position or spatial extent of a feature is also important to other domains, for example the Aviation domain, a general mechanism for tracking such features could be designed as well – which does or does not need a wrapper or transformation for providing the information required for tracking (the general Event Model could be such a wrapper).

- **Efficient Encodings** – The testbed showed that application performance with pure XML encodings is an important topic for discussion. New developments by the W3C on a binary format for generic XML already show promising evaluation results. Future work could pick up the final version of the *Efficient XML Interchange* standard and test its performance in applications using OGC standards. The results of that work would be beneficial to the whole OGC community, both application providers and their clients.

- **SWE Common specific Filter Functionality** – SWE Common uses a soft-typed approach to encode any sort of information with a set of basic data types. Operations like the filters defined by the OGC Filter Encoding Specification require operands that are encoded in a suitable way. This means that primarily GML encoded geometries (spatial or temporal) as well as simple content is used. However, an extension could be defined for the FES that specifies the rules needed to support easy spatial and temporal filtering of SWE Common encoded data. Specific functions could for example be designed which convert a SWE Common encoded position into a GML encoded geometry. This work would address an issue that has not been solved in the SWE domain so far.

- **Feature of Interest in SensorML** – Process descriptions in SensorML 1.0 do not provide information about the features that the process observes / measures. This is a gap between the SensorML and O&M model which could be closed. This would benefit the automation of integrating SensorML described sensors into Sensor Observation Services.

- **Encoding Policies** – During the testbed the need to define policies for controlling the event encoding behavior of services was identified. For example, in the aviation domain the optional boundedBy property of features needed to be available in all events sent to an Event Service so that it could support spatial filtering. In this report a different use case, that of defining that special parameters be included in observations, is discussed. Apparently there is a need for guidelines

and functionality to define policies to indicate executed, offered as well as requested behavior and include them in OGC service(s) and maybe information models. Future work could address this need.

- **Event Service Workflows** – So far, the use cases tested in OWS-7 concentrated on delivering events that were detected or derived by an Event Service to a (number of) client(s). The clients were then responsible for reacting as they see fit. To facilitate automation of workflows – like the invocation of a change detection service upon receipt of an area-of-interest entry event – the community should test the integration of event services in automated processing environments, leveraging available functionality from workflow and chaining services. Tools could also be developed or tested to facilitate this kind of integration.

- **SWE Events & Event Channels** – The work started in the OWS-6 SWE thread to define an OGC Event Architecture was continued in the OWS-7 Event Architecture cross thread. The results show that the definition of event types as well as event channels is specific to certain application domains. SWE 2.0 service specifications like the SWE Service Model or Sensor Planning Service already started with that work. However, further work in the area of eventing in Sensor Web applications should consider defining events and channels that are of common use. For example, sensor status update events would be of interest. Event channels where observations are posted that are made by certain types of sensors, that contain certain observed properties or that contain results that apply to certain geographic regions could also be beneficial (e.g. performance wise).

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 06-028r3, *OGC Sensor Alert Service Candidate Implementation Specification*

OGC 07-000, *OpenGIS® Sensor Model Language (SensorML) Implementation Specification*

OGC 07-002r3, *Observations and Measurements Part 2 - Sampling Features*

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

OGC 07-074, *OpenGIS Location Services (OpenLS): Core Services*

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

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

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

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

OASIS *Web Services Base Notification 1.3*

OASIS *Web Services Brokered Notification 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 in the OWS-6 SWE Event Architecture ER [OGC 09-032] as well as OWS-7 Event Architecture ER [10-060] shall apply.

4 Conventions

4.1 Abbreviated terms

ADT Abstract Data Type

AOI Area of Interest

CAP Common Alerting Protocol

CEP Complex Event Processing

CRS Coordinate Reference System

EML Event Pattern Markup Language

EPSG European Petroleum Survey Group Geodesy

ESB Enterprise Service Bus

ESP Event Stream Processing

EXI Efficient XML Interchange

FES Filter Encoding Specification

GML Geography Markup Language

HTTP HyperText Transfer Protocol

MUC Multi User Chat

O&M Observations & Measurements

OASIS Organization for the Advancement of Structured Information Standards

RFC Request For Comment

SAS Sensor Alert Service

SensorML Sensor Model Language

SES Sensor Event Service

SFE Sensor Fusion Enablement

SMS Short Message Service

SOS Sensor Observation Service

SPS Sensor Planning Service

SRS Spatial Reference System

SWE Sensor Web Enablement

SWES SWE Service Model

UML Unified Modeling Language

UncertML Uncertainty Markup Language

URI Unified Resource Identifier

W3C World Wide Web Consortium

WNS Web Notification Service

WS-A Web Services Addressing

WS-N Web Services Notification

WXXM Weather Information Exchange Model

XML eXtensible Markup Language

XMPP eXtensible Messaging and Presence Protocol

XPath XML Path Language

XSLT Extensible Stylesheet Language Transformations

5 Dynamic Sensor Notification – Overview

Tracking sensors and notifying users based on a geographic Area of Interest (AOI) is an important use case in many scenarios. In the context of OWS-7, tracking means receiving updates of a sensor's position. An AOI can in general be identified as a geographic point, a geographic area, a bounding box or possibly a place name / identifier.

A client should be able to subscribe at a service to automatically be notified once the presence or absence of sensors over or within an AOI is determined.

This report also covers a discussion of the relevance and relationship of standards and specifications like OASIS Common Alerting Protocol (CAP), OGC Sensor Alert Service (SAS), OGC Web Notification Service (WNS) and the OWS-6 Event Architecture.

6 Encoding of Tracked Object Position

In OWS-7, the sensors mounted on the tracked vehicle provided the following types of data:

- video stream
- video camera settings (tilt, pan, zoom)
- vehicle position (lat, lon, alt, heading, speed)
- vehicle acceleration orientation vector (x, y, z of gravity field)
- vehicle magnetic orientation vector (x, y, z of magnetic field)
- vehicle orientation vector (x, y, z in geospatial CRS)

Of these, the vehicle position is the most important information for the tracking server. It can be encoded in different ways, the advantages and disadvantages of which will be discussed in the following.

6.1 SWE Common Encoded Data Stream

Basic sensor data can be encoded using SWE Common. With 'basic' we mean data that is output directly from sensor hardware. The data structure can be described in high detail, for example providing information about the semantics of a data field, the unit of measure, as well as data quality information. Usually, the actual data is encoded following this structure and an encoding description like the swe:TextBlock encoding is used. Other encodings provide the data in binary or XML structure. The goal is to minimize the payload of a single or repeated data transmissions by sending the data description only once at the beginning of the data stream.

Note that several efforts have been made and are underway to improve the efficiency of XML data encoding, transmission and consumption. Applying a compression algorithm like zipping an XML file helps reducing the transmission size but increases the encoding and consumption time. However, a new promising effort from W3C is underway and intended to be finished in 2010: Efficient XML Interchange [1]. In fact, this is a common purpose binary format for generic XML. The performance analysis results published by the W3C working group are quite promising [2]. The aim of this new standard is to improve the overall XML process chain, from encoding and transmission to consumption. The OGC should definitely investigate the performance of this technology. A comparison of the data sizes achieved using SWE Common encodings and EXI would also be interesting.

6.1.1 Pure SWE Common Encoded Value Blocks

In this approach, the description of the sensor output (structure and used encoding) is advertised to the tracking service. After the service accepted the advertisement, encoded position data is sent to it to be matched against existing subscriptions. Matching data is sent to the subscription's consumer endpoint. The format of this notification depends on the subscription. Usually, the original data is forwarded. In some cases the data may be wrapped, transformed into a different format or fused into a new format.

The approach resembles the workflow of a Sensor Alert Service (SAS). By default, SAS requires single alert messages to be transmitted. Alert delivery is handled via XMPP but may also be performed using a Web Notification Service (WNS) – and therefore can happen via multiple protocols.

Discussion

When pure SWE Common is used to encode sensor measurements the packaging features provide an advantage. The data description is sent only once, followed by a number of efficiently encoded data blocks. The data description follows a common format and can be used to describe the outputs of a vast range of heterogeneous sensors in high detail.

A problem that arises is the fact that this approach does not leverage the O&M format, which is the default format used by a Sensor Observation Service (SOS) to encode sensor data. This is not a problem in a closed domain where one component knows the other and what that component provides. However, it is posing an interoperability issue in a general SWE environment where clients bind to new services at runtime and should therefore support the O&M model.

If SAS is used as described above then the size of a single data block is increased due to the wrapper that comes with each alert message. This is still not as big as the XML instance used in other approaches (see section 6.2).

When SAS is used to deliver alerts, messages are not self describing (as the encoded values are delivered directly). On the advertisement side this is ok as the service itself gets the data description and the alerts themselves. On the subscriber / consumer side this is different in that a consumer has to perform a DescribeAlert request (and therefore

needs to know and be able to access the SAS instance that produced an incoming alert) to get the data and encoding description. A custom way is possible in which the alert consumer is pre-configured with the data description for alerts from a certain sensor.

Using SAS means that an alert consumer is coupled to the service. It cannot handle incoming SAS alerts from previously unknown sources. This is because for understanding the alert the client needs to get the alert description. However, the alert itself does not provide the service endpoint where that description can be retrieved. As described earlier the consumer can be configured by other means with the required information. However, this means that it is no longer decoupled from the event source.

6.1.2 Multiple Output Values Encoded via DataArray in Observation Result

This approach uses a single O&M observation that has a SWE Common DataArray as its result. The array contains a list – of possibly unknown length – of measurements that are output by the sensor. Multiple outputs of the sensor are therefore grouped in a single observation, according to the timeframe requested by the client (via GetObservation). The array first describes the structure of each measurement followed by the overall encoding and then provides (a link to) the sequence of encoded values. This approach was used in SOS 1.0 implementations.

The following listing provides an example of such an observation. The actual values are provided out-of-band.

Listing 1: Observation with DataArray

```
<Observation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:gml="http://www.opengis.net/gml"
xmlns:sa="http://www.opengis.net/sampling/1.0"
xmlns:swe="http://www.opengis.net/swe/1.0.1">
  <samplingTime>
    <gml:TimePeriod>
      <gml:beginPosition>2010-04-10T21:56:00.000Z</gml:beginPosition>
      <gml:endPosition>2010-04-10T22:01:00.000Z</gml:endPosition>
    </gml:TimePeriod>
  </samplingTime>
  <procedure xlink:href="urn:ogc:object:sensor:BOTTS-INC:bottsCam0"/>
  <observedProperty xlink:href="AXIS_CAMERA_POSITION"/>
  <featureOfInterest>
    <sa:SamplingSurface gml:id="IED_AOI_POSITION">
      <sa:sampledFeature/>
      <sa:shape>
        <gml:Polygon>
          <gml:interior>
            <gml:LinearRing>
              <gml:posList>34.73610510862753 -86.7411185718009
                                              34.73601919338803 -86.73358035517214
                                              34.73361510917333 -86.72805797951885
                                              34.73957640913068 -86.722679290518
                                              34.7443799628379  -86.72659132721732
                                              34.74474637313518 -86.74109642336798
                                              34.73610510862753 -
86.7411185718009</gml:posList>
            </gml:LinearRing>
          </gml:interior>
        </gml:Polygon>
      </sa:shape>
    </sa:SamplingSurface>
  </featureOfInterest>
  <result>
```



```
    <swe:DataArray>
      <swe:elementCount>
        <swe:Count>
          <swe:value>0</swe:value>
        </swe:Count>
      </swe:elementCount>
      <swe:elementType name="Position">
        <swe:DataRecord definition="urn:ogc:def:property:OGC::dynamicLocation">
          <swe:field name="systemTime">
            <swe:Time definition="urn:ogc:def:property:OGC::samplingTime"
referenceTime="1970-01-01T00:00:00.000Z" gml:id="SYSTEM_CLOCK">
              <swe:uom code="ms"/>
            </swe:Time>
          </swe:field>
          <swe:field name="location">
            <swe:Vector referenceFrame="urn:ogc:def:crs:EPSG::5329">
              <swe:coordinate name="latitude">
                <swe:Quantity definition="urn:ogc:def:property:OGC::latitude"
gml:id="LAT">
                  <gml:description>The latitude component of location</gml:description>
                  <swe:uom code="deg"/>
                  <swe:constraint>
                    <swe:AllowedValues>
                      <swe:interval>-90.0 90.0</swe:interval>
                    </swe:AllowedValues>
                  </swe:constraint>
                </swe:Quantity>
              </swe:coordinate>
              <swe:coordinate name="longitude">
                <swe:Quantity definition="urn:ogc:def:property:OGC::longitude"
gml:id="LON">
                  <gml:description>The longitude component of location</gml:description>
                  <swe:uom code="deg"/>
                  <swe:constraint>
                    <swe:AllowedValues>
                      <swe:interval>-180.0 180.0</swe:interval>
                    </swe:AllowedValues>
                  </swe:constraint>
                </swe:Quantity>
              </swe:coordinate>
              <swe:coordinate name="altitude">
                <swe:Quantity definition="urn:ogc:def:property:OGC::altitude"
gml:id="ALT">
                  <gml:description>The altitude component of location</gml:description>
                  <swe:uom code="m"/>
                </swe:Quantity>
              </swe:coordinate>
            </swe:Vector>
          </swe:field>
          <swe:field name="speed">
            <swe:Quantity definition="urn:ogc:def:property:OGC::speed" gml:id="SPEED">
              <gml:description>The magnitude of velocity in the forward
direction</gml:description>
              <swe:uom code="m/s"/>
            </swe:Quantity>
          </swe:field>
          <swe:field name="direction">
            <swe:Quantity definition="urn:ogc:def:property:OGC::trueHeading"
gml:id="DIRECTION">
              <gml:description>The true heading direction of the
platform</gml:description>
              <swe:uom code="deg"/>
            </swe:Quantity>
          </swe:field>
          <swe:field name="accuracy">
            <swe:Quantity definition="urn:ogc:def:property:OGC::accuracy"
gml:id="ACCURACY">
              <gml:description>The accuracy of the GPS location</gml:description>
              <swe:uom code="m"/>
            </swe:Quantity>
          </swe:field>
          <swe:field name="providerType">
```

```
            <swe:Category definition="urn:ogc:def:property:OGC::sensorType"/>
          </swe:field>
        </swe:DataRecord>
      </swe:elementType>
      <swe:encoding>
        <swe:TextBlock decimalSeparator="." blockSeparator="@@" tokenSeparator=","/>
      </swe:encoding>
      <swe:values>1270936561000,34.7435599565506,-
86.73452854156494,212.0,8.0,343.125,4.0,gps@@1270936562000,34.743618965148926,-
86.73456072807312,212.0,7.75,341.01563,4.0,gps@@1270936563000,34.74368333816528,-
86.73458218574524,211.0,7.75,341.71875,4.0,gps@@ … </swe:values>
    </swe:DataArray>
  </result>
</Observation>
```

Discussion

The advantage of this approach is that multiple data blocks from one of a sensor's outputs (there may be multiple, see SensorML - OGC 07-000) are provided in an efficient encoding. Metadata – like the feature of interest and the data description – that does not change is only provided once.

That the data stored in an SOS's database could be structured into one observation according to the timeframe requested by the client was also deemed advantageous.

However, this only works well with SWE Common encoded data. A generic SOS cannot perform such aggregation for all its observations because of the difficulties explained in the following.

One reason is that the result type in observations can differ. For example, if a client requested position data from a set of sensors then it may very well be the case that this data is encoded in GML geometries, like a GML Point. However, the sensor position can also be encoded using a SWE Common data record. Both approaches may be used by an SOS's sensors. In that case the service is not able to provide all the resulting data in just one observation. A client would therefore need to request data from one sensor only and also know in advance that this sensor encodes the data in SWE Common.

When a GetObservation request targets more than one sensor one could argue that multiple observations from one sensor can still be aggregated in a single observation if the sensor uses SWE Common to encode the results and that such observations then become part of the GetObservation response. This is doable so long as observation metadata like the result time or the quality does not change for the aggregated observations. If it did then the service would need to split the aggregating observation into multiple parts.

Again one could argue that all observation information, like the sampling time as well as result time and quality are part of the sensors output and should therefore be part of an observations result value. However, by doing so one ignores the purpose of using O&M to encode sensor data and devalues the model to a simple container format. The benefit of using O&M is that it explicitly models the relationship between a process, a property that it observes / measures, the feature that property belongs to, the temporal context of the observation and the result of the observation act. In addition, information about the

circumstances of this "sampling event" can be provided, like the parameters under which the observation was made as well as quality factors (like uncertainty) of the observation act. The model enables clients to create filter statements to query for observations whose properties have specific relationships and values. If this model is not followed then clients need to 1) know in which ways it is not followed and 2) create custom filter statements. The information needed for 1) is not provided by a service yet and at the moment can only be achieved by inspecting a given observation.

If clients were not able to use explicit filter statements then a workaround would seem to be the usage of imprecise filters like "sensor position within area of interest". This is nice for clients but forces the service to have built-in knowledge about the data structures used. This is impossible for a tracking server that consumes generic observations and hard for a tracking server that consumes observations with aggregated SWE Common data. The problem for the latter is that additional constraints and rules need to be followed to actually identify the correct position information (i.e. the quantities that contain the coordinates that provide the sensor position). This primarily requires precise semantic definitions of the components in the SWE Common data.

When the O&M model is ignored as discussed above one could also not use it at all. This would be contrary to the SWE architecture and more akin to defining a specific purpose GML application schema. The latter is a valid approach. It has for example been applied in the WXXM model to specialize the general O&M model to contain specific result types and observed properties. A similar specialization to provide sensor position information can be investigated for the tracking service in the future as well (in OWS-7 a different approach was tested – see section 7.2).

Note: aggregating the measurements from a sensor into a new observation is a valid approach. However, the procedure referenced in this new observation should not be the sensor that generated the aggregated measurements but rather describe the aggregation process itself.

Another problem with this kind of aggregating observations is that it is difficult for a service to cancel or revise such observations if errors are detected later on. For example, someone recognizes that the result values of one or more of the aggregated observations was wrong and therefore wants to revise them. In order to inform clients like the tracking service that received the erroneous observation about the revision, the observation needs to be identifiable. This means that the event source which created the aggregating observation needs to store that observation and also information on which single measurements are part of it. Now, if one or more of the single measurements is revised the event source can identify which aggregating observations (keep in mind that single measurements can be part of multiple aggregating observations) are affected and revise them accordingly. It would be easier to simply revise the single measurements and thus send them directly to the tracking service.

6.2 O&M Observation

An observation provides information about a single output value of a sensor. It relates this result value to the observed property, the feature the property belongs to, the process that generated the value and quality of the observation act and value. Additional

properties of the observation for example provide information about parameters of the observation event.

The reason to investigate pure observations as defined by O&M (OGC 07-022r1) to provide the position information is to find a generic approach for tracking sensors that works with all sensor observations.

6.2.1 Position as Observation Result

Sensors like a GPS measure position information. Usually a point location is provided in a certain SRS – for example EPSG:4979. In addition orientation data like true heading can be provided. Observations from such sensors provide result values for some sort of position property.

Discussion

The observation's result type depends upon the type of the feature of interest used in the observation. If no specific feature type is available to be used in the observation then sampling features as defined in OGC 07-002r3 can be used. They serve as intermediaries that provide the relationships between the base observation and the results of all further processing performed with that observation (see Listing 2).

Listing 2: Observation with SamplingPoint and SWE Common encoded result

```
<om:Observation xmlns:om="http://www.opengis.net/om/1.0"
xmlns:gml="http://www.opengis.net/gml" xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:sa="http://www.opengis.net/sampling/1.0">
  <om:samplingTime>
    <gml:TimeInstant>
      <gml:timePosition>2010-04-05T15:59:00+02:00</gml:timePosition>
    </gml:TimeInstant>
  </om:samplingTime>
  <om:procedure xlink:href="MY-SML-Described-SYSTEM"/>
  <om:observedProperty xlink:href="urn:ogc:def:property:OGC::dynamicLocation"/>
  <om:featureOfInterest>
    <sa:SamplingPoint>
      <sa:sampledFeature xlink:href="urn:ogc:def:nil:OGC:unknown"/>
      <sa:position>
        <gml:Point>
          <gml:pos srsName="urn:ogc:def:crs:EPSG:7.4.1:4979">34.739429354667664 -
86.72852575778961 239.0</gml:pos>
        </gml:Point>
      </sa:position>
    </sa:SamplingPoint>
  </om:featureOfInterest>
  <om:result>
    <swe:DataRecord definition="urn:ogc:def:property:OGC::dynamicLocation">
      <swe:field name="systemTime">
        <swe:Time definition="urn:ogc:def:property:OGC::samplingTime"
referenceTime="1970-01-01T00:00:00.000Z" gml:id="SYSTEM_CLOCK">
          <swe:uom code="ms"/>
          <swe:value>1270475940</swe:value>
        </swe:Time>
      </swe:field>
      <swe:field name="location">
        <swe:Vector referenceFrame="urn:ogc:def:crs:EPSG:7.4.1:4979">
          <swe:coordinate name="latitude">
            <swe:Quantity definition="urn:ogc:def:property:OGC::latitude" gml:id="LAT">
              <gml:description>The latitude component of location</gml:description>
```



```
              <swe:uom code="deg"/>
              <swe:constraint>
                <swe:AllowedValues>
                  <swe:interval>-90.0 90.0</swe:interval>
                </swe:AllowedValues>
              </swe:constraint>
              <swe:value>34.739429354667664</swe:value>
            </swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="longitude">
            <swe:Quantity definition="urn:ogc:def:property:OGC::longitude" gml:id="LON">
              <gml:description>The longitude component of location</gml:description>
              <swe:uom code="deg"/>
              <swe:constraint>
                <swe:AllowedValues>
                  <swe:interval>-180.0 180.0</swe:interval>
                </swe:AllowedValues>
              </swe:constraint>
              <swe:value>-86.72852575778961</swe:value>
            </swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="altitude">
            <swe:Quantity definition="urn:ogc:def:property:OGC::altitude" gml:id="ALT">
              <gml:description>The altitude component of location</gml:description>
              <swe:uom code="m"/>
              <swe:value>239.0</swe:value>
            </swe:Quantity>
          </swe:coordinate>
        </swe:Vector>
      </swe:field>
      <swe:field name="speed">
        <swe:Quantity definition="urn:ogc:def:property:OGC::speed" gml:id="SPEED">
          <gml:description>The magnitude of velocity in the forward
direction</gml:description>
          <swe:uom code="m/s"/>
          <swe:value>0.0</swe:value>
        </swe:Quantity>
      </swe:field>
      <swe:field name="direction">
        <swe:Quantity definition="urn:ogc:def:property:OGC::trueHeading"
gml:id="DIRECTION">
          <gml:description>The true heading direction of the platform</gml:description>
          <swe:uom code="deg"/>
          <swe:value>127.265625</swe:value>
        </swe:Quantity>
      </swe:field>
      <swe:field name="accuracy">
        <swe:Quantity definition="urn:ogc:def:property:OGC::accuracy" gml:id="ACCURACY">
          <gml:description>The accuracy of the GPS location</gml:description>
          <swe:uom code="m"/>
          <swe:value>48.0</swe:value>
        </swe:Quantity>
      </swe:field>
      <swe:field name="providerType">
        <swe:Category definition="urn:ogc:def:property:OGC::sensorType">
          <swe:value>gps</swe:value>
        </swe:Category>
      </swe:field>
    </swe:DataRecord>
  </om:result>
</om:Observation>
```

In case that the property of a known feature is observed the result type of the observation needs to match accordingly. The following listing shows a GeometryObservation which has a gml:Point as result.

Listing 3: GeometryObservation providing the sensor position

```
<omx:GeometryObservation xmlns:omx="http://www.opengis.net/omx/1.0"
xmlns:om="http://www.opengis.net/om/1.0" xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <om:samplingTime>
    <gml:TimeInstant>
      <gml:timePosition>2009-09-08T15:59:00+02:00</gml:timePosition>
    </gml:TimeInstant>
  </om:samplingTime>
  <om:procedure xlink:href="MY-GPS"/>
  <om:observedProperty xlink:href="http://my.domain/FeatureType/Vehicle/position"/>
  <om:featureOfInterest xlink:href="http://my.org/vehicle.xml"/>
  <om:result>
    <gml:Point>
      <gml:pos srsName="urn:ogc:def:crs:EPSG:6.14:4326">40.85 -74.0608</gml:pos>
    </gml:Point>
  </om:result>
</omx:GeometryObservation>
```

As a GeometryObservation only specializes the generic observation to have a geometry as result, it may not be specific enough for the tracking use case which expects single point measures. A further specialization could be performed.

In any case, note that the value of the observedProperty together with the value of the feature of interest determines the allowed type of the observation's result. As it is possible to encode the position data of features in various ways the observations providing values for the position may use results of different types as well, which makes a tracking service hard to realize if it uses the result value of observation's for tracking.

Another issue with this approach is that the identification of the feature given in the feature of interest can be difficult. Think about a system of sensors that is attached to a vehicle. One of the sensor components is a GPS. We have different options to populate the feature of interest used in observations from this sensor. On the one hand it can be the vehicle itself, modeled in some domain specific application schema. On the other hand it can be the sensor system itself. If clients only know the identifier of the vehicle and use that identifier in a tracking subscription to identify relevant observations then they will never get a notification in case that the GPS measurements use the sensor system as the feature of interest. Thus clients need to be able to determine the relationships between sensor processes and the features these sensors make observations for. This information is not directly available in the observations sent to the tracking service so it would need additional information. For example, a SensorML description could be provided when registering a new event source with the tracking service. However, in SensorML 1.0 process descriptions do not provide information about the observed or observable features of interest.

6.2.2 Position via FeatureOfInterest

The approach described in section 6.2.1 looks for the position of a tracked entity in the result of an observation. A different approach is to look for the position in an observation's feature of interest.

Discussion

Here, the observed property may be anything. It may for example be a temperature value or image, even a combination of different phenomena. In this approach the tracking service investigates the observation's feature of interest to determine the position of the tracked entity. If, for example, the feature of interest is a SamplingPoint (like in Listing 2) the service would use that location as the location of the tracked entity. This approach was tested in OWS-7 (see section 7.2). The problem with this approach is that it only works with in-situ measurements, where the location of the feature of interest and the procedure in an observation are more or less the same. Another problem that was identified by the SWE community is that arbitrary feature types may be used as an observation's feature of interest. The issue for the tracking service is that it – or rather the clients that create the tracking subscriptions – would need to understand all the feature types used in incoming notifications and be able to identify which of their properties contain the required position data. For clients it is impossible to take into account an unlimited set of feature types when subscribing.

Another way to look at this issue is to search for the position of the tracked entity in the procedure information given with an observation. SensorML encoded procedure data may provide position information if the sensor is a physical one (see OGC 07-000). This can be a single position, vector or dynamic process. However, the way how to encode a list of time dependent positions is not well defined (yet). Also, SensorML descriptions may be given in varying degrees of detail so that the position information may not be available at all.

In any case, a similar issue as the one described before arises. First of all the procedure descriptions in O&M observations may be given in an unlimited set of encodings, only one of which is SensorML. Second, this approach also assumes that in-situ measurement is performed. In a remote sensing environment the locations of the sensor and the feature that is observed may be quite different – think of a satellite that observes a vehicle on earth driving on a road.

At least some solution for the issue of arbitrary feature / procedure types may be solved soon as described in section 6.2.3.

6.2.3 Position as Observation Parameter

Finding the spatial position in one of the properties of an observation's feature of interest or procedure can become very difficult because of the multitude of possible encodings (see discussion in section 6.2.2). Special observation parameters can provide the required information directly.

Discussion

This approach is the result of discussions in the SWE community. Instead of requiring clients to search for the position property in an observation's feature of interest or procedure, the entity that produces the observation adds special observation parameters that contain the according property values. The following listing shows how this works in

an O&M 2.0 observation (note that version 2.0 of O&M was not available at the time when this report was written).

Listing 4: Draft O&M 2.0 observation with sampling geometry parameter

```
<om:Observation xmlns:om="http://www.opengis.net/om/x-2.0"
xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:swe="http://www.opengis.net/swe/2.0" gml:id="oc95">
  <om:samplingTime>
    <gml:TimeInstant>
      <gml:timePosition>2009-09-08T15:59:00+02:00</gml:timePosition>
    </gml:TimeInstant>
  </om:samplingTime>
  <om:procedure xlink:href="MY-HWS:0e1ad0a7-e2a0-479f-b05a-1ea38b2269e8"/>
  <om:observedProperty xlink:href="urn:ogc:def:property:OGC::x-surfacewatertemperature"/>
  <om:featureOfInterest
xlink:href="http://my.wfs.de?request=getFeature&amp;featureid=z27s67f"/>
  <om:parameter>
    <om:NamedValue>
      <om:name>urn:ogc:def:x-conformance-class:OGC::sos:ext:sg:1.0</om:name>
      <om:value>
        <gml:Point>
          <gml:pos srsName="urn:ogc:def:crs:EPSG:7.4:4326">52.0 8.67</gml:pos>
        </gml:Point>
      </om:value>
    </om:NamedValue>
  </om:parameter>
  <om:result xsi:type="swe:QuantityPropertyType">
    <swe:Quantity definition="urn:ogc:def:property:OGC::x-surfacewatertemperature">
      <swe:uom code="Cel"/>
      <swe:value>13.7</swe:value>
    </swe:Quantity>
  </om:result>
</om:Observation>
```

As we can see the name of a NamedValue parameter in an observation defines its semantics. It also serves as a code to identify the expected value type. This can be used to provide values both of the sensor and / or feature of interest position at the sampling time of the observation. A tracking service could then leverage the according parameter values.

The drawback here is that this approach is not defined for O&M 1.0 and would not be realizable as shown in the listing (because an O&M 1.0 observation does not foresee parameters with arbitrarily encoded values).

In addition there is no requirement defined by O&M that these parameters be included in observations. So there is no guarantee per se that observations sent to a tracking service provide the required information. However, what we can investigate in the future is the use of service policies to indicate that these properties have to be included in the observation events sent from event sources. In addition – if the input side of a tracking service is irrelevant – policies / capability statements to indicate that all observations published by the event service contain the required information can be designed.

6.3 Domain Specific Application Schema

Many domains create their own models to capture and exchange the information that is relevant for satisfying their use cases. GML Application Schema are a suitable and (in the

OGC) well-known way to create a UML model of the feature and data types of a given domain and map that model to an XML Schema representation that defines the structure for information exchange. Sometimes such models serve as kind of meta-models to capture basic information and relationships that is useful in several domains; O&M is one such model. These models can then be specialized further to satisfy more specific requirements.

6.3.1 Observation Specialization

As discussed in section 6.2.1, the observation types defined by O&M are quite generic and are difficult to use for the purpose of tracking an entity. The generic observation type could be specialized to restrict the observation properties and solve the problems that arise through unrestricted property type domains.

Discussion

This approach would allow constraining the values of an observation's observed property and result to fit the needs of a tracking service. However, this approach requires that entities that need to be tracked or sensors observing them generate these observations. This may or may not be feasible. In addition, it would require that the observed property is part of the feature of interest and that the property's type fits. This requirement cannot be fulfilled for all feature types. The fallback is to require that a sampling feature is used as the feature of interest, which itself points to the ultimate feature for which the position is provided. Through this additional constraint at least queries involving the identity of the ultimate feature and its position can be made.

6.3.2 Special Feature Type

For the purpose of providing the information required for tracking an entity, a GML Application Schema can be designed.

Discussion

Whenever a specific set of information is needed to satisfy the requirements of a given domain, a new application schema – in other words a new UML package – defining appropriate types can be designed. This is the way to go if no existing model fully satisfies the requirements. This also means that the new application schema may depend on one or more other models or is designed to be used in extension points defined in these models.

The new schema can define features, types and data types to encode a single position together with any additional information required for tracking. It can also provide support for the delivery of position lists / tracks. Such lists can be used for storing position information and performing queries on them. They can also be used for batch transmissions of multiple positions to a client, for example an event service.

The drawback of this approach is that it is not directly tied to sensor observations, thus a pre-processing step is required to generate the new information out of sensor observations.

In light of this discussion, it is useful to take a look at the Location Service (OpenLS) standards defined by the OGC. In version 1.2 of these standards, a single Position ADT (type) is defined, which provides information about a mobile terminal's location. It is reported as "*an observation/calculated position for a mobile terminal, but can be any position used by the platform. Contains Point with optional Shape, QoP, Speed, Direction and Time. Also has levelOfConfidence attribute*." This position can be queried through a Gateway service. Version 1.2 of the OpenLS standard provides a table of use cases for the Gateway service that indicates that a mobile terminal's position can also be pushed to a client and that a "triggered location" may be available. However, the latter is not explained in the specification, presumably because it has priority three which is said to be optional. That a position can also be pushed to a client may mean that a client can subscribe to be notified of a terminal's position. However, it seems more likely that it is some form of asynchronous response that can be requested by the client. The "responseType" parameter of a Gateway service request with value "PUSH" indicates this – however, that parameter is not available in the 1.2 schema of the OpenLS XLS message with SLIR body. Thus the OpenLS services do not seem to support tracking functionality as required in OWS-7.

6.4 Summary

This section provided an overview of multiple approaches how sensor position used for tracking can be encoded. Characteristics as well as advantages and disadvantages of each approach have been discussed.

In summary, two general approaches can be identified. On the one hand position can be provided through sensor observations directly. On the other hand a specific model is used to provide the required information.

Using sensor observations directly has several issues that make an easy use of generic observations for tracking purposes difficult. Differing position encodings, differing locations where to put the position and differing options where to look for the feature that shall actually be tracked are the primary issues to be solved. Solutions for these issues have been discussed. A holistic approach incorporating all these solutions may be investigated in the future.

An easier approach seems to be to agree on a specific information model to use for tracking. Although a pre-processing step would be required to transform sensor observation data into the types defined by that model, it would move the burden of understanding the diversity of possible observation structures from the tracking service and client to the entity that generates the observations – and which should therefore understand what the observation is about. Such a model together with the tracking workflow – transforming sensor observations to the common model, publishing the encoded position data to the tracking service, subscribing for it and ultimately notifying clients accordingly – should be the focus of future work.

The encoding efficiency of position data used by a tracking service is a different concern. As outlined in section 6.1, SWE Common provides an efficient way of encoding multiple

positions. However, new approaches exist that are designed to provide a general way to efficiently encode and handle XML data (Efficient XML Interchange, see section 6.1).

A different aspect that should be investigated in the future as well is how good SWE Common encoded data structures fit into the model assumed by the OGC Filter Encoding Specification. For example, does an application that supports the FES also support filtering of the tuples in a SWE Common DataArray? What is the output of a filtered DataArray? This aspect has only been touched slightly in OWS-7 but the SWE community as a whole would benefit from a thorough investigation.

7 Implementations

7.1 SAS Based Implementation

7.1.1 Introduction

Compusult's Tracking Notification/Event Service implementation is based on the Sensor Alert Service (SAS) Version: 0.9, an OGC Best Practice document. This implementation builds on much of the work Compusult completed in OWS-6. The defined use case for event notification required determining when a moving observation point (truck coordinates) intersected a line (archived video track). The functionality provided by a SAS-based event service was determined applicable. The addition of a SOAP wrapper and the inclusion of CAP as an alert notification format were necessary extensions to meet the requirements.

The Event Service operations are as follows:

- Advertise
- GetCapabilities
- DescribeSensor
- DescribeAlert
- Subscribe
- CancelSubscription
- RenewSubscription

Advertise

The SAS specification provides an Advertise operation that allows services to be registered with the Event Service. In the use case, this operation was used to register a Sensor Observation Service for each moving truck (i.e. a mobile sensor) and a WPS service that is used for change detection notification.

DescribeSensor and DescribeAlert

The Advertise Request, sent to the Event Service, provides the information necessary for the DescribeSensor and DescribeAlert operations. The Advertise request denotes the format of the response returned from the DescribeAlert operation. The DescribeAlert is used to decode the alert notifications that are pushed to the Event Service by the sensor.

GetCapabilities

The GetCapabilities operation provides the information (originally retrieved and parsed from the Advertise operation) needed for a client to Subscribe to an advertised sensor. For this project, two services were advertised by the Event Service and displayed via the Capabilities document - a Mobile Sensor and the WPS.

Subscribe

The subscription XML request contains the identifier of the mobile sensor to follow and a vector of coordinates to test the sensor's current location against.

The Map Client subscribes to all the mobile sensors to receive any alert (i.e. no filter criteria set) sent out by the sensor. This will allow the map client to initiate an action if necessary when an alert is received (i.e. indicate the point the track intersected etc.). Upon subscription submission by a user, the Event Service Client records relevant information (linked to the returned subscription id and user contact id) to use when an alert notification is picked up by the Map Client. In this case, the information would be the archived track metadata. This archived data coupled with the mobile sensor metadata returned with the alert notification could be then used to initiate an Event Service subscription to the WPS for change detection.

CancelSubscription and RenewSubscription

These operations provide the ability to suspend a subscription and reactivate at a later date. In the case of tracking a mobile sensor's location, the benefits of these operations may be negligible. The alert will not be triggered if the mobile sensor is not in the defined area and, in the scenario, we would want to receive any alert determined. Therefore, there is no obvious benefit, to cancel and/or renew the subscription, as opposed to leaving it active.

7.1.2 Workflow

The workflow for the event service would be as shown in Figure 1 to Figure 3.

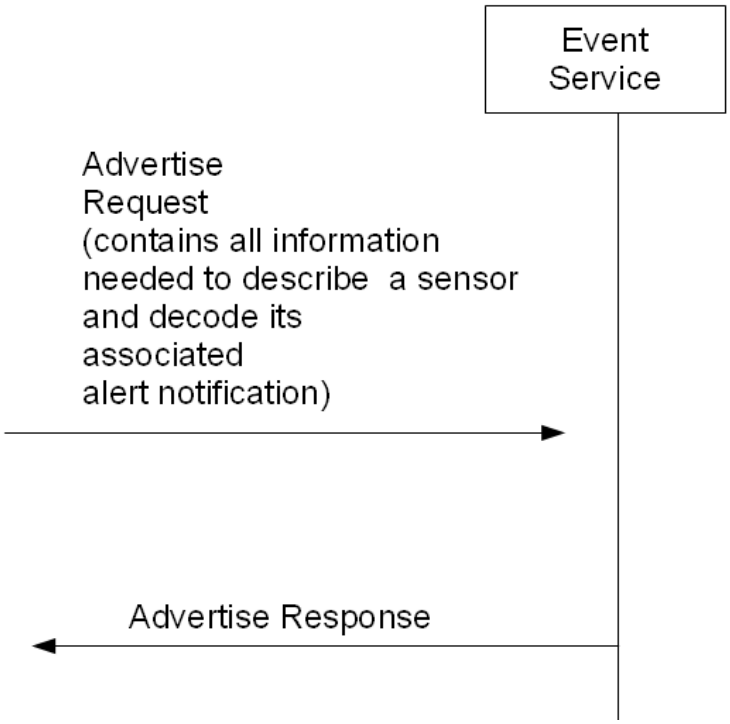

Figure 1: SAS based implementation workflow – advertising the event source

An Advertise request for sensor(s) is submitted and accepted by the Event Service. The subscriptions offerings are indicated by the Event Service's GetCapabilities response.

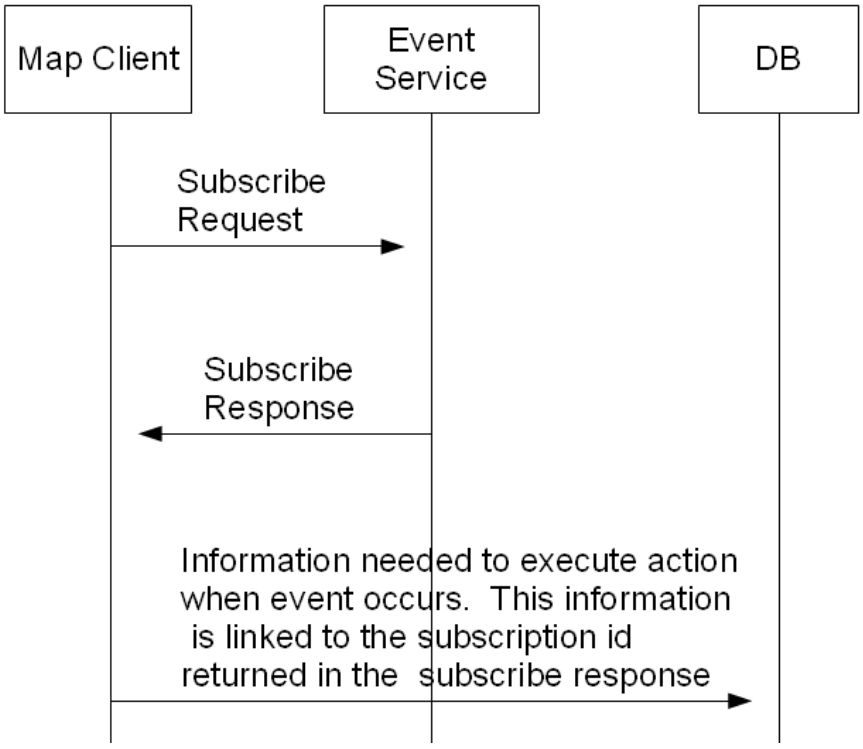

Figure 2: SAS based implementation workflow – subscribing to the event service

A client submits a Subscribe request for an advertised sensor to the Event Service.

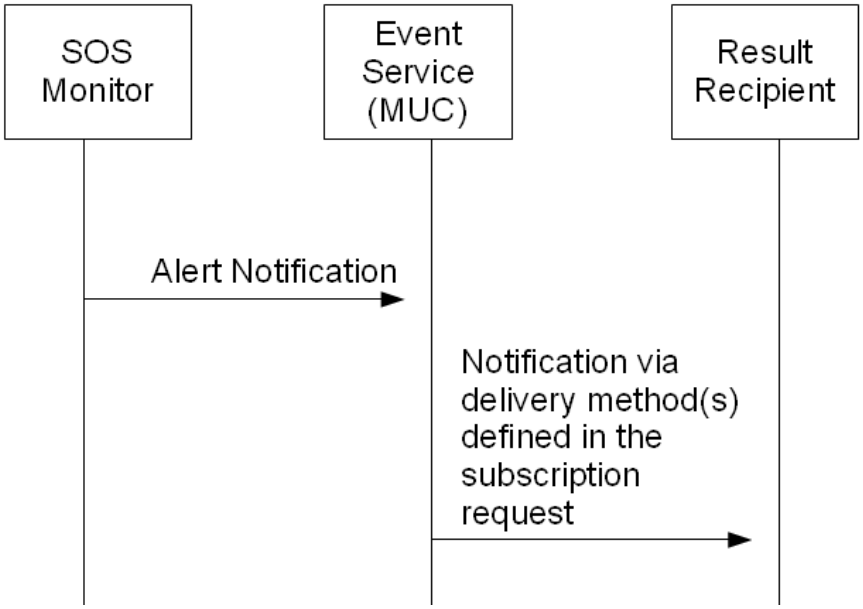

Figure 3: SAS based implementation workflow – event publication

The sensor pushes an alert notification to the Event Service. The Event Service is listening on the port assigned to the sensor and receives the notification. The service checks all subscriptions to the Sensor. If criteria are matched, alert notification is formatted as indicated by the subscriptionFormat and sent to the resultRecipient indicated. If the resultRecipient is the MapClient, then the client executes a unique action using relevant information from the database.

Figure 4 to Figure 7 provide a more elaborate view upon the different parts of the workflow.

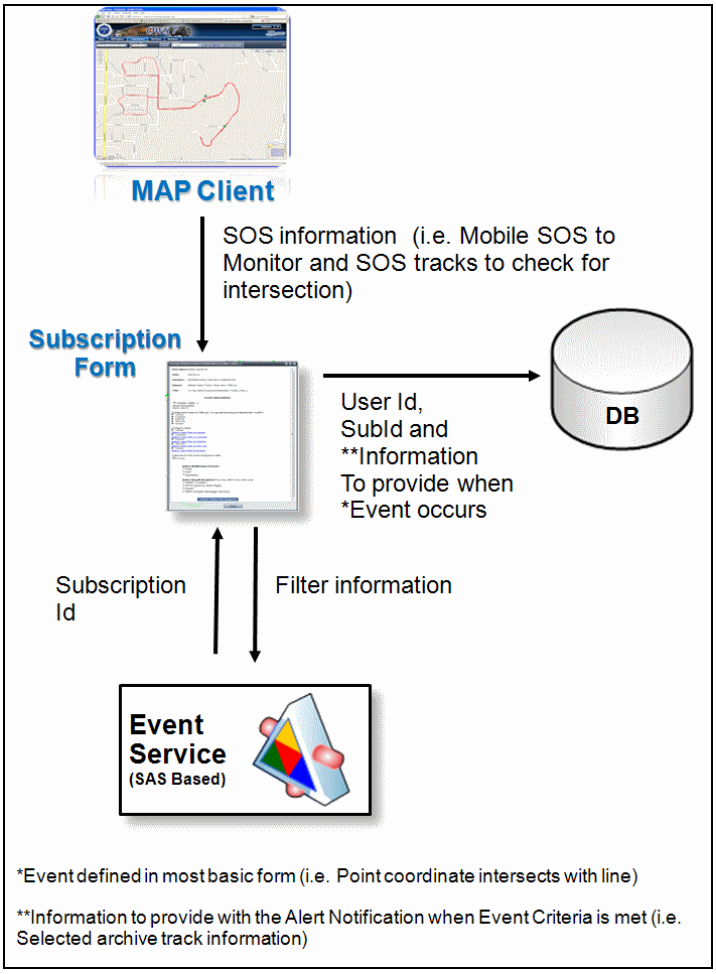

Figure 4: SAS based implementation workflow – elaboration part 1

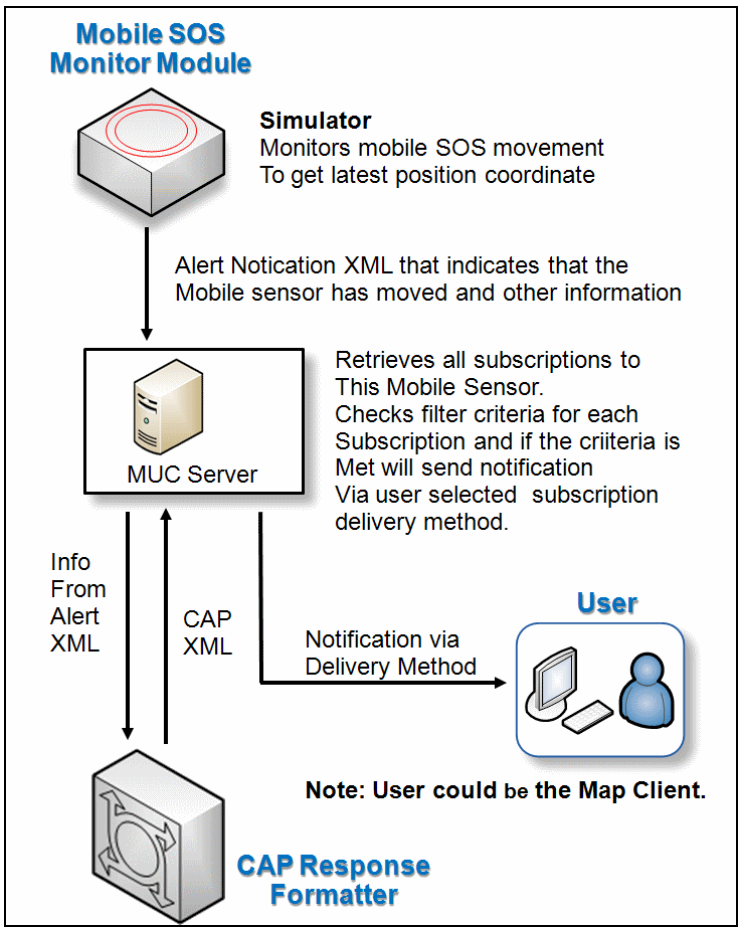

Figure 5: : SAS based implementation workflow – elaboration part 2

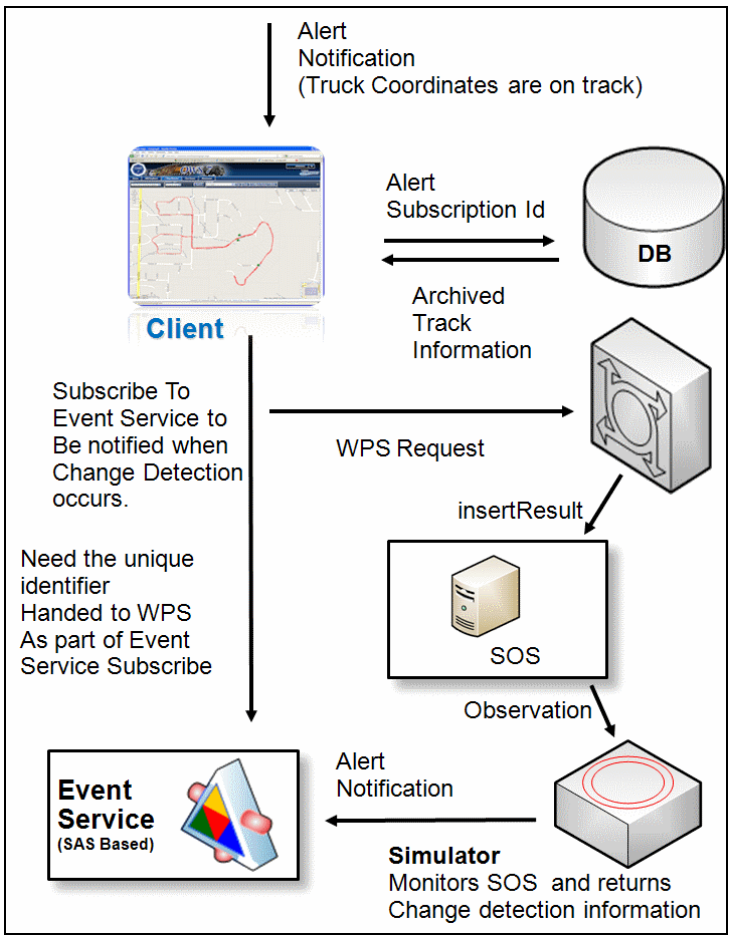

Figure 6: SAS based implementation workflow – elaboration part 3

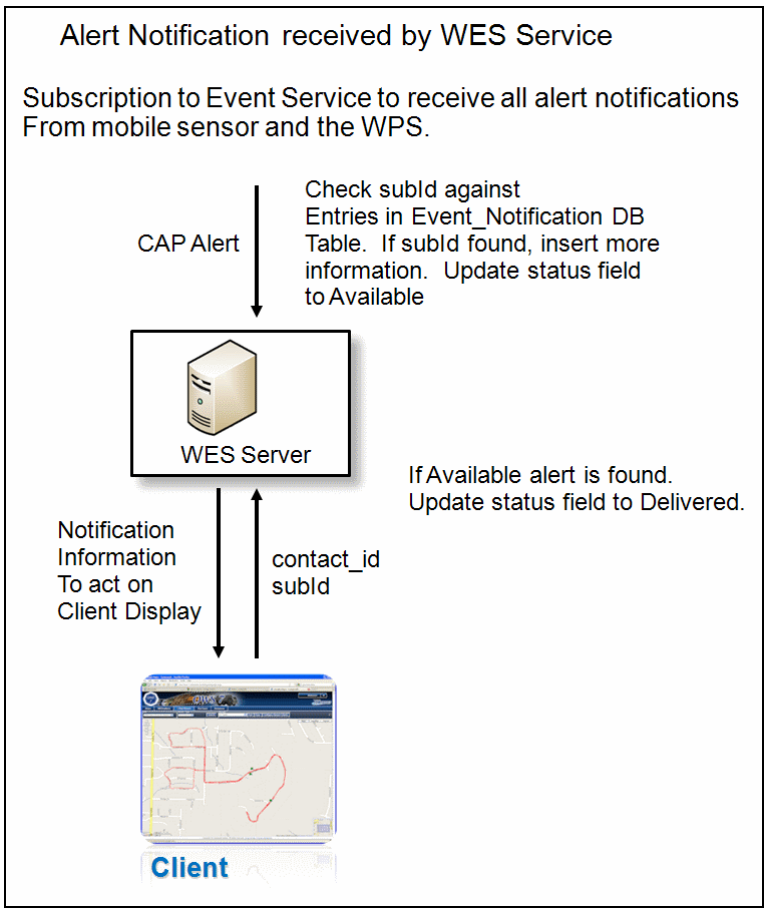

Figure 7: SAS based implementation workflow – elaboration part 4

7.1.2.1 Publishing Position Events to the Service

The DescribeAlert response is used to decode the alert notifications that are sent to the Event Service. For position events, there is currently a prerequisite that the observation properties include latitude and longitude.

Listing 5: advertise operation for mobile_video_1

```
<Advertise xmlns="http://www.opengis.net/sas/0.0"
xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:ows="http://www.opengis.net/ows"
xmlns:wns="http://www.opengis.net/wns/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
service="SAS" version="0.9.0">
<messageStructure>
<swe:DataBlockDefinition>
<swe:components name="urn:ogc:def:phenomenon:MobileVideo::ALERTS">
<swe:DataRecord>
<swe:field name="Latitude">
<swe:Quantity
definition="urn:ogc:def:phenomenon:MobileVideo::ALERTS:Latitude"/>
</swe:field>
<swe:field name="Longitude">
<swe:Quantity
definition="urn:ogc:def:phenomenon:MobileVideo::ALERTS:Longitude"/>
</swe:field>
<swe:field name="Direction">
<swe:Quantity
definition="urn:ogc:def:phenomenon:MobileVideo::ALERTS:Direction"/>
</swe:field>
<swe:field name="IMG_URL">
<swe:Quantity
definition="urn:ogc:def:phenomenon:MobileVideo::ALERTS:IMG_URL"/>
</swe:field>
<swe:field name="Duration">
<swe:Quantity
definition="urn:ogc:def:phenomenon:MobileVideo::ALERTS:Duration"/>
</swe:field>
</swe:DataRecord>
</swe:components>
<swe:encoding>
<swe:TextBlock decimalSeparator="." blockSeparator=" " tokenSeparator=
"$$"/>
</swe:encoding>
</swe:DataBlockDefinition>
</messageStructure>
<sensorDescription>urn:ogc:def:procedure:MobileVideo::mobile_video_1</se
nsorDescription>
</Advertise>
```

Listing 6: Advertise request for SOS urn:ogc:procedure:BottsCam_2010_04_09

```
<Advertise xmlns="http://www.opengis.net/sas/0.0"
xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:ows="http://www.opengis.net/ows"
xmlns:wns="http://www.opengis.net/wns/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="SAS"
version="0.9.0"><messageStructure><swe:DataBlockDefinition><swe:componen
ts
name="urn:ogc:def:phenomenon:MobileVideo::ALERTS"><swe:DataRecord><swe:f
ield name="CameraSetting_Pan"><swe:Quantity
definition="urn:ogc:def:phenomenon:CameraSetting_Pan"/></swe:field><swe:
field name="CameraSetting_Tilt"><swe:Quantity
definition="urn:ogc:def:phenomenon:CameraSetting_Tilt"/></swe:field><swe
:field name="CameraSetting_Zoom"><swe:Quantity
definition="urn:ogc:def:phenomenon:CameraSetting_Zoom"/></swe:field><swe
:field name="Location_Accuracy"><swe:Quantity
definition="urn:ogc:def:phenomenon:Location_Accuracy"/></swe:field><swe:
field name="Location_Altitude"><swe:Quantity
definition="urn:ogc:def:phenomenon:Location_Altitude"/></swe:field><swe:
field name="Location_Direction"><swe:Quantity
definition="urn:ogc:def:phenomenon:Location_Direction"/></swe:field><swe
:field name="Location_Latitude"><swe:Quantity
definition="urn:ogc:def:phenomenon:Location_Latitude"/></swe:field><swe:
field name="Location_Longitude"><swe:Quantity
definition="urn:ogc:def:phenomenon:Location_Longitude"/></swe:field><swe
:field name="Location_ProviderType"><swe:Quantity
definition="urn:ogc:def:phenomenon:Location_ProviderType"/></swe:field><
swe:field name="Location_Speed"><swe:Quantity
definition="urn:ogc:def:phenomenon:Location_Speed"/></swe:field><swe:fie
ld name="Orientation_Accuracy"><swe:Quantity
definition="urn:ogc:def:phenomenon:Orientation_Accuracy"/></swe:field><s
we:field name="Orientation_Azimuth"><swe:Quantity
definition="urn:ogc:def:phenomenon:Orientation_Azimuth"/></swe:field><sw
e:field name="Orientation_Pitch"><swe:Quantity
definition="urn:ogc:def:phenomenon:Orientation_Pitch"/></swe:field><swe:
field name="Orientation_Roll"><swe:Quantity
definition="urn:ogc:def:phenomenon:Orientation_Roll"/></swe:field><swe:f
ield name="Video_Frame_URL"><swe:Quantity
definition="urn:ogc:def:phenomenon:Video_Frame_URL"/></swe:field></swe:D
ataRecord></swe:components>
<swe:encoding>
<swe:TextBlock decimalSeparator="." blockSeparator=" " tokenSeparator=
"$$"/>
</swe:encoding>
</swe:DataBlockDefinition>
</messageStructure>
<sensorDescription>urn:ogc:procedure:BottsCam_2010_04_09</sensorDescript
ion>
</Advertise>
```

Listing 7: Advertise request for WPS

```
<Advertise xmlns="http://www.opengis.net/sas/0.0"
xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:ows="http://www.opengis.net/ows"
xmlns:wns="http://www.opengis.net/wns/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="SAS"
version="0.9.0">
<messageStructure>
<swe:DataBlockDefinition>
<swe:components name="urn:ogc:def:phenomenon:WPS::ALERTS">
<swe:DataRecord>
<swe:field name="Identifier">
<swe:Quantity
definition="urn:ogc:def:phenomenon:WPS::ALERTS:Identifier"/>
</swe:field>
<swe:field name="Title">
<swe:Quantity definition="urn:ogc:def:phenomenon:WPS::ALERTS:Title"/>
</swe:field>
<swe:field name="DataReference">
<swe:Quantity
definition="urn:ogc:def:phenomenon:WPS::ALERTS:DataReference"/>
</swe:field>
</swe:DataRecord>
</swe:components>
<swe:encoding>
<swe:TextBlock decimalSeparator="." blockSeparator=" " tokenSeparator=
"$$"/>
</swe:encoding>
</swe:DataBlockDefinition>
</messageStructure>
<sensorDescription>WPS</sensorDescription>
</Advertise>
```

7.1.2.2 Creating a Subscription at the Service

The SAS specification, for Subscribe operation, provides the ability for a vector of coordinates to be handed in and valueFilter(s) to be defined against the sensor's observed properties. This vector of coordinates can be tested against the current location of the mobile sensor. Each pair of sequential coordinates of the provided point vector is tested against the location coordinate provided by the mobile sensor. If the location coordinate is between two points on the line, an alert notification is sent to the appropriate result recipients.

Functionality to provide valueFilters (i.e. isEqual, isBetween etc...) against each observed property is available but not necessary for this particular use case.

Listing 8: Subscription for mobile video location

```
<?xml version="1.0" encoding="UTF-8"?>
<Subscribe xmlns="http://www.opengis.net/sas/0.0"
xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:ows="http://www.opengis.net/ows"
xmlns:wns="http://www.opengis.net/wns/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="SAS"
version="0.9.0">
<sas:SensorID>urn:ogc:def:procedure:MobileVideo::mobile_video_1</sas:Sen
sorID>
<sas:EventFilter>
<sas:ValueFilterList>
<sas:Location>
<swe:Position definition="1-Track">
<swe:location>
<swe:Vector>
<swe:coordinate>
<swe:Quantity definition="Latitude"><swe:uom
code="deg"/><swe:value>35.21</swe:value></swe:Quantity>
</swe:coordinate>
<swe:coordinate>
<swe:Quantity definition="Longitude"><swe:uom code="deg"/><swe:value>-
87.33</swe:value></swe:Quantity>
</swe:coordinate>
<swe:coordinate>
<swe:Quantity definition="Latitude"><swe:uom
code="deg"/><swe:value>34.99</swe:value></swe:Quantity>
</swe:coordinate>
<swe:coordinate>
<swe:Quantity definition="Longitude"><swe:uom code="deg"/><swe:value>-
87.11</swe:value></swe:Quantity>
</swe:coordinate>
</swe:Vector>
</swe:location>
</swe:Position>
</sas:Location>
</sas:ValueFilterList>
</sas:EventFilter>
<sas:ResultRecipient>
<wns:NotificationTarget>
<wns:NotificationChannel>
<wns:Email>paula@compusult.net</wns:Email>
</wns:NotificationChannel>
</wns:NotificationTarget>
</sas:ResultRecipient>
</Subscribe>
```

The Subscribe response provides a unique subscription id. This id can be used to renew or cancel the subscription. According to the SAS specification, the subscribe response would also return a XMPP URI, however, for the use case, the mobile sensor (truck) is a SOS that is being polled (via getLatestObservation) and a Multi-User Chat (MUC) implementation is used internally but is not exposed.

Listing 9: Response for mobile video subscription

```
<?xml version="1.0" encoding="UTF-8"?> <SubscribeResponse
SubscriptionID="sub1681" expires="2014-01-01T00:00:Z"
xsi:schemaLocation="http://www.opengis.net/sas/0.0../sasSubscribe.xsd"
xmlns="http://www.opengis.net/sas/0.0"
xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:ows="http://www.opengis.net/ows"
xmlns:wns="http://www.opengis.net/wns/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="SAS"
version="0.9.0">
<sas:AlertChannel>
<sas:XMPPURI/>
</sas:AlertChannel>
</SubscribeResponse>
```

The alert message that would be pushed to the service for mobile_video_1 is shown in Listing 10.

Listing 10: Example alert message pushed to the service for mobile_video_1

```
<?xml version="1.0" encoding="UTF-8"?>
<NotificationMessage xmlns="http://www.opengis.net/wns/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<ServiceDescription>
<ServiceType>SAS</ServiceType>
<ServiceTypeVersion>0.9.0</ServiceTypeVersion>
<ServiceURL>http://ows-7.compusult.net/EventService/SAS</ServiceURL>
</ServiceDescription>
<Payload>
<Alert>
<SensorID>urn:ogc:def:procedure:MobileVideo::mobile_video_1</SensorID>
<Timestamp>2010-05-05T00:00:00Z</Timestamp>
<AlertData>34.77$$-86.75$$NW$$http://ows-
7.compusult.net/wes/SOSClient/TrackBuilder/DemoTrackImages/image1.jpg$$5
</AlertData>
</Alert>
</Payload>
</NotificationMessage>
```

7.1.2.3 Notification of Client

Notification of alert is sent in CAP. For a mobile sensor, the CAP XML would indicate the line coordinates given in the subscription and the observed property values associated with the sensor when the criteria were met. This should provide the information necessary to initiate further action when received by the resultRecipient.

Listing 11: Sample CAP alert

```
<?xml version=\"1.0\" encoding =\"UTF-8\" ?>
<alert xmlns=\"urn:oasis:names:tc:emergency:cap:1.1\">
<identifier>sub01</identifier>
<sender>Alert received from sensorId </sender>
<sent>2010-05-25T16:49:00-0700</sent>
<status>Actual</status>
<msgType>Alert</msgType>
<scope>Restricted</scope>
<restricted>Subscribers to OWS-7 Event Service</restricted>
<info>
<restriction>To receive this alert message, subscription to the Event
Service is required.</restriction>
<category>Other</category>
<event>Subscription criteria has been met.  Please see description to
review observed properties.</event>
<urgency>Immediate</urgency>
<severity>Unknown</severity>
<certainty>Observed</certainty>
<description>Sensor observed properties and associated
values</description>
<resource>
<resourceDesc>Please access ows-7.compusult.net to initiate further
action</resourceDesc>
<uri>http://ows-7.compusult.net</uri>
</resource>
<area>
<polygon>coordinates in line intersected</polygon>
</area>
</info>
</alert>
```

7.1.3 Summary

There were advantages and disadvantages of using the SAS as the basis for the Event Service. The advantages of the SAS are in its simplicity. The format of the alert messages pushed to the Event Service are short and concise. It provides a smooth mechanism to define a simple event. The simplicity of the alert notification data (a string split by predefined tokens) sent to this Event Service is a significant advantage when considering limitations of sending information across the web (i.e. bandwidth). Sending an alert notification when a point intersects a line falls easily into events definable by the SAS specification.

However, the simplicity of the SAS also has some notable disadvantages. One significant limitation of the current SAS specification is that the subscription invokes an implicit **and** between filter criteria which, in our case, forced a subscription request per track as opposed to a single subscription request for a number of archived tracks. The addition of an **or** concept would make the Subscribe request implementation more efficient. In our case, it would allow a single subscription for multiple tracks.

This service implementation puts the onus on the receiving resultRecipient (as opposed to the service itself) to execute an action upon alert receipt. It is likely that different clients would want to perform different actions depending on information returned by a sensor.

This implementation minimizes overhead by not incorporating this information in the service. The receiving resultRecipient is responsible for piecing together simple events to execute a unique action.

7.2 WS-Notification Based Implementation

7.2.1 Introduction

The Tracking and Notification Service provided by the Institute for Geoinformatics (IfGI) of the University of Münster is based on the Sensor Event Service (SES). The SES specification is available as an OGC discussion paper (08-133). One design goal of the SES was to make more use of existing standards than the Sensor Alert Service (SAS) does. Thus, the communication was switched to implement OASIS Web Services Notification (WS-N) with SOAP via HTTP replacing the XMPP based notification mechanism of the SAS and the operations common in the publish-subscribe messaging pattern.

The service implementation is based on the SES prototype developed by the IfGI and 52° North. This prototype was already successfully used in the OWS-6 AIM Thread as Event Service. Besides the use in SFE, the prototype was also enhanced and extended in the OWS-7 Aviation thread.

7.2.2 Service Operations

In this section the available service operations are described. At first the general interaction is presented followed by a description of the most important operations.

7.2.2.1 Overview

Figure 8 gives an overview of the important operations of the Tracking and Notification Service. It contains five components: The service itself, a Registrar which can perform registrations of Publishers, a Publisher which sends notifications to the service, a Subscriber which is able to subscribe Consumers for notifications and a Consumer which is able to receive notifications. Note that these components can be merged. For instance the Subscriber and the Consumer could be implemented in a single client application like it is done in the OWS-7 SFE thread. The next sections describe the operations in more detail.

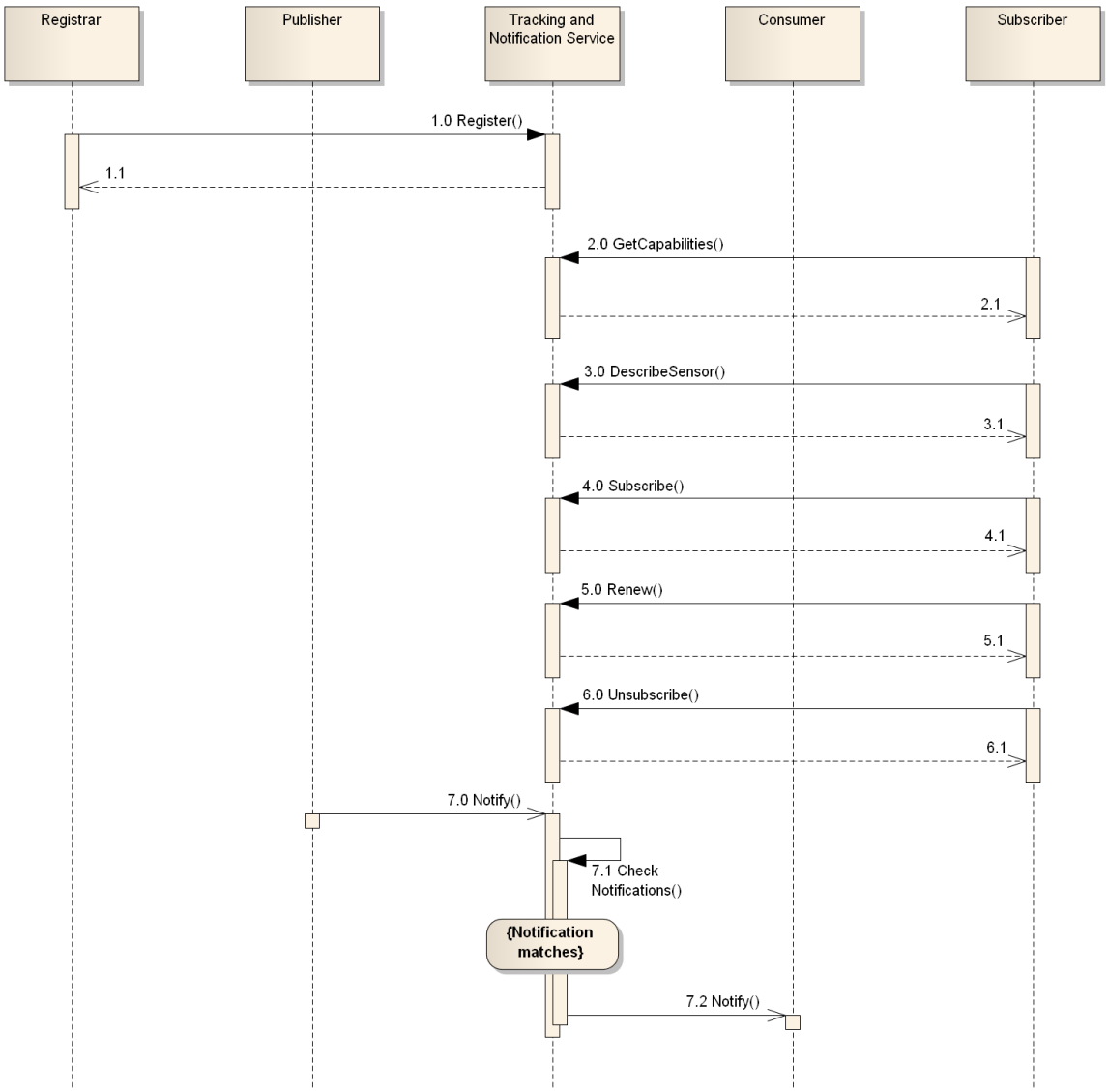

Figure 8: Overview of the important service operations

7.2.2.2 RegisterPublisher

The Register operation is used to make a publisher known to the Tracking and Notification Service. This operation is inherited from OASIS WS-Notification and is mandatory to implement if a brokering functionality shall be used at the service. However, the method is not mandatory to be used to allow unknown Publishers as event source. As the payload of this operation contains a SensorML description it is also sometimes not obvious to register a Publisher, especially if it is not a sensor. In the OWS-7 SFE scenario the Publisher is an extended SOS service which is not registered at the Tracking and Notification Service. A registration can be removed via the DestroyRegistration operation.

7.2.2.3 GetCapabilities and DescribeSensor

The GetCapabilities operation returns the service metadata and its capabilities. The response describes for instance the supported filter languages and lists the registered Publishers. The DescribeSensor operation returns details about a registered Publisher by returning the stored SensorML document.

7.2.2.4 Subscribe

With this operation one can express the interest in specific notifications. It is also inherited from WS-N. The request contains the information where notifications shall be sent to. Also a restriction which subset of notifications shall be sent can be included like a filter statement or a specific topic.

7.2.2.5 Renew and Unsubscribe

These two WS-N methods allow managing subscriptions. With Renew one can extend the time until when a subscription is active. Via Unsubscribe, subscriptions can be removed before they run out of time.

7.2.2.6 Notify

This method is used by WS-N to actually send notifications. It has to be implemented by all notification consumers. In the case of the OWS-7 scenario this is the Tracking and Notification Service on the one hand, receiving notifications from the video SOS and the client on the other hand receiving notifications on close video tracks by the Tracking and Notification Service. In contrast to the other operations this is no request but a one way message from a Publisher to a Consumer.

7.2.2.7 GetCurrentMessage

This operation is also defined by WS-N. It allows requesting the last notification that was sent via a specific topic. The concept behind it is to allow newly subscribed consumers to access the last message they missed before their subscription. All following messages should be received only via the Notify method.

7.2.3 Workflow in OWS-7

The workflow regarding the IfGI Tracking and Notification Service in the OWS-7 scenario looks as follows (see Figure 9):

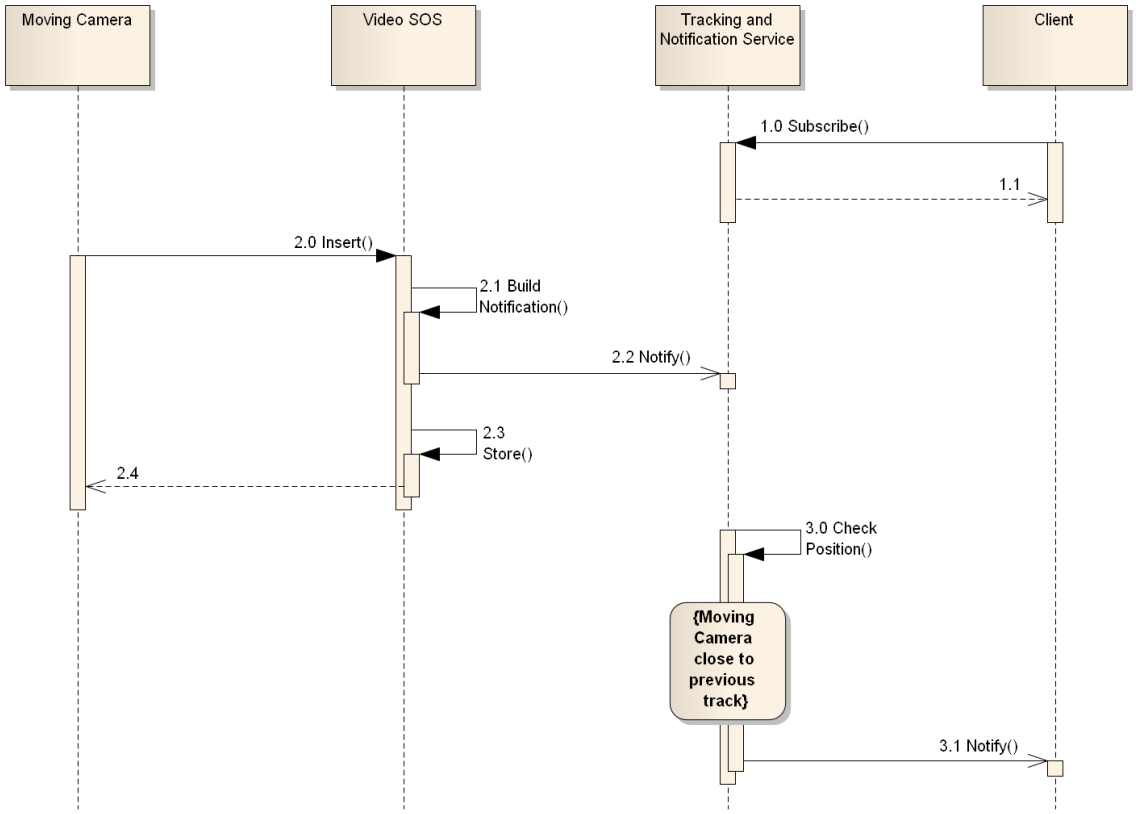

Figure 9 - Tracking and Notification workflow

There are three workflow steps that interact with the Tracking and Notification Service. At first the client subscribes at the service. Within the subscription instructions are given in which cases the service shall send notifications to the Client. The subscription is described in section 7.2.3.1 in more detail.

The next interaction takes place if new data is inserted into the Video SOS. Besides storing it the SOS also builds a notification for the Tracking and Notification Service containing the newly entered data. The notifications are described in section 7.2.3.2.

After receiving notifications from the Video SOS the Tracking and Notification Service executes the instructions provided via the client's subscription. If the position of the moving camera went close to a previous recorded track a notification is sent to the client. These notifications are described in section 7.2.3.3.

7.2.3.1 Creating a Subscription at the Service

The subscribe request sent by the client looks as shown in the following listing.

Listing 12: Example Subscribe request

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   xmlns:wsa="http://www.w3.org/2005/08/addressing"
   xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
  <soap:Header>
   <wsa:To>http://v-tml.uni-muenster.de:8080/ses
                        /services/SesPortType</wsa:To>
   <wsa:Action>http://docs.oasis-open.org/wsn/bw-
2/NotificationProducer/SubscribeRequest</wsa:Action>
   <wsa:MessageID>uuid:4e595160-185a-9b3c-3eb6-
592c7c5b0c7a</wsa:MessageID>
   <wsa:From>
    <wsa:Address>http://www.w3.org/2005/08/addressing
                        /role/anonymous</wsa:Address>
   </wsa:From>
  </soap:Header>

  <soap:Body>
   <wsnt:Subscribe>
    <wsnt:ConsumerReference>
     <wsa:Address>%Client_Address%</wsa:Address>
    </wsnt:ConsumerReference>

    <wsnt:Filter>
     <wsnt:MessageContent
      Dialect="http://www.opengis.net/ses/filter/level3">
      <eml:EML xmlns:eml="http://www.opengis.net/eml/0.0.1"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:fes="http://www.opengis.net/fes/2.0"
         xmlns:gml32="http://www.opengis.net/gml/3.2">
        <eml:SimplePatterns>
         <!-- get positions that are within a distance to the given
          geometry -->
         <eml:SimplePattern inputName="input"
patternID="withinPattern">
          <eml:SelectFunctions>
           <eml:SelectFunction newEventName="withinEvent">
            <eml:SelectEvent eventName="input"/>
           </eml:SelectFunction>
          </eml:SelectFunctions>
          <eml:View>
           <eml:LengthView>
            <eml:EventCount>1</eml:EventCount>
           </eml:LengthView>
          </eml:View>
          <eml:Guard>
           <fes:Filter>
            <fes:DWithin>
             <fes:ValueReference>
              input/geometry
             </fes:ValueReference>
             <gml32:Polygon gml32:id="polygon_01"
                 srsName="urn:ogc:def:crs:EPSG:7.4.1:4326">
              <gml32:exterior>
               <gml32:LinearRing>
                <gml32:coordinates decimal="." cs="," ts=" ">
```

```
                    %Previous_Track%
                  </gml32:coordinates>
                </gml32:LinearRing>
              </gml32:exterior>
            </gml32:Polygon>
            <fes:Distance uom="m">100</fes:Distance>
          </fes:DWithin>
        </fes:Filter>
      </eml:Guard>
      <eml:PropertyRestrictions/>
    </eml:SimplePattern>
    <!-- get positions that are not within a distance to the given
      geometry -->
    <eml:SimplePattern inputName="input"
patternID="outsidePattern">
      <eml:SelectFunctions>
        <eml:SelectFunction newEventName="outsideEvent">
          <eml:SelectEvent eventName="input"/>
        </eml:SelectFunction>
      </eml:SelectFunctions>
      <eml:View>
        <eml:LengthView>
          <eml:EventCount>1</eml:EventCount>
        </eml:LengthView>
      </eml:View>
      <eml:Guard>
        <fes:Filter>
          <fes:Not>
            <fes:DWithin>
              <fes:ValueReference>
                input/geometry
              </fes:ValueReference>
              <gml32:Polygon gml32:id="polygon_02"
                 srsName="urn:ogc:def:crs:EPSG:7.4.1:4326">
                <gml32:exterior>
                  <gml32:LinearRing>
                    <gml32:coordinates decimal="." cs="," ts=" ">
                      %Previous_Track%
                    </gml32:coordinates>
                  </gml32:LinearRing>
                </gml32:exterior>
              </gml32:Polygon>
              <fes:Distance uom="m">100</fes:Distance>
            </fes:DWithin>
          </fes:Not>
        </fes:Filter>
      </eml:Guard>
      <eml:PropertyRestrictions/>
    </eml:SimplePattern>
  </eml:SimplePatterns>
  <eml:ComplexPatterns>
    <!-- get positions outside that are fllowed by a position
inside
      the geometry as result -->
    <eml:ComplexPattern patternID="entrancePattern">
      <eml:SelectFunctions>
        <eml:SelectFunction newEventName="entranceEvent"
           createCausality="true" outputName="out">
```



```
              <eml:NotifyOnSelect>
                <eml:Message>AOI entered</eml:Message>
              </eml:NotifyOnSelect>
            </eml:SelectFunction>
          </eml:SelectFunctions>
          <eml:StructuralOperator>
            <eml:BEFORE/>
          </eml:StructuralOperator>
          <eml:FirstPattern>
            <eml:PatternReference>outsidePattern</eml:PatternReference>
            <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
          </eml:FirstPattern>
          <eml:SecondPattern>
            <eml:PatternReference>withinPattern</eml:PatternReference>
            <eml:SelectFunctionNumber>0</eml:SelectFunctionNumber>
          </eml:SecondPattern>
        </eml:ComplexPattern>
      </eml:ComplexPatterns>
      <eml:TimerPatterns/>
      <eml:RepetitivePatterns/>
    </eml:EML>
  </wsnt:MessageContent>
 </wsnt:Filter>
</wsnt:Subscribe>
</soap:Body>
</soap:Envelope>
```

The most important parts in this request are the information in the SOAP header, the consumer reference and the filter statement. In the SOAP header the operation (wsa:Action element) and the target endpoint of the operation (wsa:To element) are specified. The consumer reference and the filter statement are located in the SOAP body.

The consumer reference (wsnt:ConsumerReference element) specifies the address where notifications shall be sent to. This has to be a Web Services Addressing compliant endpoint on which the WS-Notification Notify operation is implemented.

The largest part of the subscribe request contains the filter statement (wsnt:Filter element). It contains usually filtering instructions in some filter language like for instance XPath or the OGC Filter Encoding. In the case of the SES however, it is possible to include processing instructions to perform Complex Event Processing (CEP) operations. Via CEP it is possible not only to filter incoming notifications but also to derive new information from them by matching them against so called event patterns. In the OWS-7 SFE scenario the capabilities of a CEP enabled Event Service are demonstrated.

The event patterns matched against the camera data are encoded using the Event Pattern Markup Language (EML) which is available as an OGC Discussion Paper (08-132). Its use is indicated in the filter statement by the message content dialect set to SES filter level 3 (http://www.opengis.net/ses/filter/level3). More information on the filter levels can be found in the SES specification (08-133).

The enclosed EML document defines three event patterns that are executed. The first two are so called simple patterns which perform filtering operations. The results of these patterns are then combined in a so called complex pattern which derives information from the incoming events and generates the output (see Figure 10).

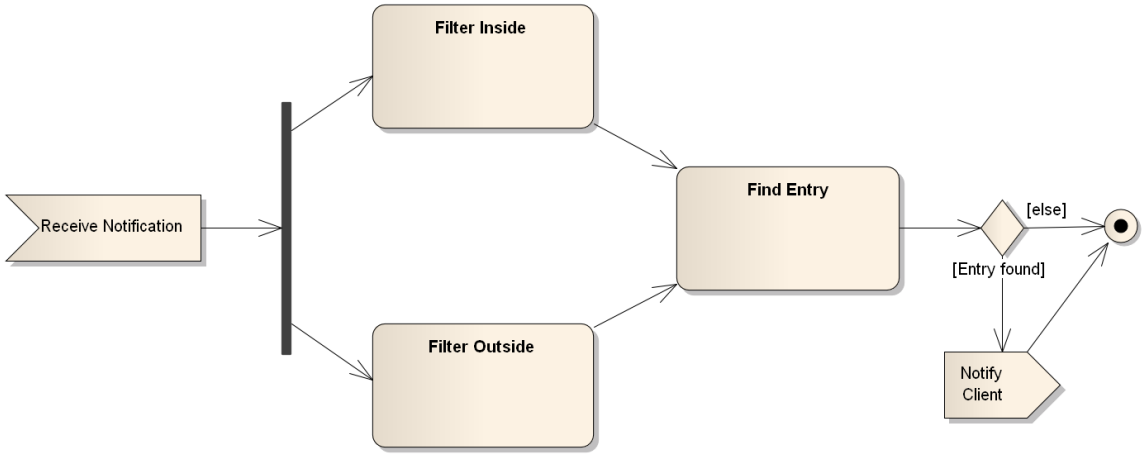

Figure 10 - Overview of the EML event patterns

The two filtering patterns check if the moving camera is close to a previously recorded track. The camera position is given within the notification (see section 7.2.3.2). The filters consist of a Filter Encoding statement using the "DWithin" operator with the distance for the buffer and the geometry to match against as parameters. In order to separate the positions that are not close to the previous track from those close to the track the "DWithin" filter is inverted with a "Not" operator in one case. The two event streams are inserted in the final pattern that checks if an event away from the track is followed by an event close to the track. In this case the camera entered the region around the previously recorded track.

By performing these three steps instead of just filtering if a position is close to a previous track there is only one alert sent to the consumer. Unnecessary notifications that would happen otherwise are avoided.

7.2.3.2 Publishing Events to the Service

In this section the content of the notifications sent to the Tracking and notification service in step 2.2 (see Figure 9) is described. Basically this step is a Notify request as defined by WS-Notification. The following listing shows an example request.

Listing 13: Example input for the WS-N based Tracking and Notification service

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
xmlns:om="http://www.opengis.net/om/1.0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
```



```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:sa="http://www.opengis.net/sampling/1.0">
 <soap:Header>
  <wsa:To>http://v-tml.uni-muenster.de:8080/ses
                       /services/SesPortType</wsa:To>
  <wsa:Action>http://docs.oasis-open.org/wsn/bw-2
                       /NotificationConsumer/Notify</wsa:Action>
  <wsa:MessageID>uuid:1b4d3025-f80a-a5b6-aa37-
864c47fa1a7e</wsa:MessageID>
  <wsa:From>
   <wsa:Address>
    http://www.w3.org/2005/08/addressing/role/anonymous
   </wsa:Address>
  </wsa:From>
 </soap:Header>

 <soap:Body>
  <wsnt:Notify>
   <wsnt:NotificationMessage>
    <wsnt:Message>

     <om:Observation>
      <om:samplingTime>
       <gml:TimeInstant>
        <gml:timePosition>2010-04-
05T15:59:00+02:00</gml:timePosition>
       </gml:TimeInstant>
      </om:samplingTime>
      <om:procedure xlink:href="MY-SML-Described-SYSTEM"/>
      <om:observedProperty xlink:href=
                        "urn:ogc:def:property:OGC::dynamicLocation"/>
      <om:featureOfInterest>
       <sa:SamplingPoint>
        <sa:sampledFeature
xlink:href="urn:ogc:def:nil:OGC:unknown"/>
        <sa:position>
         <gml:Point>
          <gml:pos srsName="urn:ogc:def:crs:EPSG:7.4.1:4979">
           14.739429354667664 -86.72852575778961 239.0
          </gml:pos>
         </gml:Point>
        </sa:position>
       </sa:SamplingPoint>
      </om:featureOfInterest>

      <om:result>
       <swe:DataRecord definition=
                        "urn:ogc:def:property:OGC::dynamicLocation">
        <swe:field name="systemTime">
         <swe:Time definition=
                        "urn:ogc:def:property:OGC::samplingTime"
                   referenceTime="1970-01-01T00:00:00.000Z"
                   gml:id="SYSTEM_CLOCK">
          <swe:uom code="ms"/>
          <swe:value>1270475940</swe:value>
         </swe:Time>
        </swe:field>
```

```
            <swe:field name="location">
              <swe:Vector
referenceFrame="urn:ogc:def:crs:EPSG:7.4.1:4979">
                <swe:coordinate name="latitude">
                  <swe:Quantity definition=
                              "urn:ogc:def:property:OGC::latitude"
                              gml:id="LAT">
                    <gml:description>
                      The latitude component of location
                    </gml:description>
                    <swe:uom code="deg"/>
                    <swe:constraint>
                      <swe:AllowedValues>
                        <swe:interval>-90.0 90.0</swe:interval>
                      </swe:AllowedValues>
                    </swe:constraint>
                    <swe:value>34.739429354667664</swe:value>
                  </swe:Quantity>
                </swe:coordinate>
                <swe:coordinate name="longitude">
                  <swe:Quantity definition=
                              "urn:ogc:def:property:OGC::longitude"
                              gml:id="LON">
                    <gml:description>
                      The longitude component of location
                    </gml:description>
                    <swe:uom code="deg"/>
                    <swe:constraint>
                      <swe:AllowedValues>
                        <swe:interval>-180.0 180.0</swe:interval>
                      </swe:AllowedValues>
                    </swe:constraint>
                    <swe:value>-86.72852575778961</swe:value>
                  </swe:Quantity>
                </swe:coordinate>
                <swe:coordinate name="altitude">
                  <swe:Quantity definition=
                              "urn:ogc:def:property:OGC::altitude"
                              gml:id="ALT">
                    <gml:description>
                      The altitude component of location
                    </gml:description>
                    <swe:uom code="m"/>
                    <swe:value>239.0</swe:value>
                  </swe:Quantity>
                </swe:coordinate>
              </swe:Vector>
            </swe:field>
            <swe:field name="speed">
              <swe:Quantity definition="urn:ogc:def:property:OGC::speed"
                              gml:id="SPEED">
                <gml:description>
                  The magnitude of velocity in the forward direction
                </gml:description>
                <swe:uom code="m/s"/>
                <swe:value>0.0</swe:value>
              </swe:Quantity>
            </swe:field>
```



```
            <swe:field name="direction">
              <swe:Quantity definition=
                            "urn:ogc:def:property:OGC::trueHeading"
                            gml:id="DIRECTION">
                <gml:description>
                  The true heading direction of the platform
                </gml:description>
                <swe:uom code="deg"/>
                <swe:value>127.265625</swe:value>
              </swe:Quantity>
            </swe:field>
            <swe:field name="accuracy">
              <swe:Quantity
definition="urn:ogc:def:property:OGC::accuracy"
                            gml:id="ACCURACY">
                <gml:description>
                  The accuracy of the GPS location
                </gml:description>
                <swe:uom code="m"/>
                <swe:value>48.0</swe:value>
              </swe:Quantity>
            </swe:field>
            <swe:field name="providerType">
              <swe:Category definition=
                            "urn:ogc:def:property:OGC::sensorType">
                <swe:value>gps</swe:value>
              </swe:Category>
            </swe:field>
          </swe:DataRecord>
        </om:result>
      </om:Observation>
    </wsnt:Message>
  </wsnt:NotificationMessage>
 </wsnt:Notify>
</soap:Body>
</soap:Envelope>
```

The payload of the notification message is an OGC O&M Observation. It contains the information necessary for filtering included as sampling time and feature of interest. The result contains a SWE Common data record which contains the full information of the event including the current speed, heading and GPS accuracy.

An O&M Observation was chosen as data encoding to provide clearly defined elements that contain the event time and location which is important for the definition of temporal and spatial filter statements. SWE Common data records do not support this through their generic layout. There, filter statements can only be built if the specific structure of the data record is known in advance. Section 6 describes this problem in more detail.

7.2.3.3 Notification of the Client

The following listing shows a shortened example of the output of the Tracking and Notification service. Similar as the input it uses the WS-Notification Notify operation. For a better readability; here only the message payload is shown.

Listing 14: Example of the output of the WS-N based Tracking and Notification service

```
<DerivedEvent xmlns="http://www.opengis.net/em/0.2.0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:ns="http://www.opengis.net/om/1.0"
xmlns:ns1="http://www.opengis.net/sampling/1.0"
xmlns:ns2="http://www.opengis.net/swe/1.0.1"
xmlns:xlink="http://www.w3.org/1999/xlink">
 <eventTime>
  <gml:TimePeriod>
   <gml:beginPosition>2010-04-05T15:59:11+02:00</gml:beginPosition>
   <gml:endPosition>2010-04-05T15:59:11+02:00</gml:endPosition>
  </gml:TimePeriod>
 </eventTime>

 <attribute>
  <NamedValue>
   <name>value</name>
   <value>AOI entered</value>
  </NamedValue>
 </attribute>

 <member>
  <EventEventRelationship>
   <role>http://www.opengis.net/em/roles/0.2/cause</role>
   <target>
    <ns:Observation>
     <ns:samplingTime>
      <gml:TimeInstant>
       <gml:timePosition>2010-04-05T15:59:00+02:00</gml:timePosition>
      </gml:TimeInstant>
     </ns:samplingTime>
     <ns:procedure xlink:href="MY-SML-Described-SYSTEM"/>
     <ns:observedProperty

 xlink:href="urn:ogc:def:property:OGC::dynamicLocation"/>
     <ns:featureOfInterest>
      <ns1:SamplingPoint>
       <ns1:sampledFeature xlink:href="urn:ogc:def:nil:OGC:unknown"/>
       <ns1:position>
        <gml:Point>
         <gml:pos srsName="urn:ogc:def:crs:EPSG:7.4.1:4979">
          14.739429354667664 -86.72852575778961 239.0
         </gml:pos>
        </gml:Point>
       </ns1:position>
      </ns1:SamplingPoint>
     </ns:featureOfInterest>
     <ns:result>
      <ns2:DataRecord definition=
                        "urn:ogc:def:property:OGC::dynamicLocation">
       […]
      </ns2:DataRecord>
     </ns:result>
    </ns:Observation>
   </target>
  </EventEventRelationship>
 </member>
```

```
<member>
  <EventEventRelationship>
    <role>http://www.opengis.net/em/roles/0.2/cause</role>
    <target>
      <ns:Observation>
        <ns:samplingTime>
          <gml:TimeInstant>
            <gml:timePosition>2010-04-05T15:59:10+02:00</gml:timePosition>
          </gml:TimeInstant>
        </ns:samplingTime>
        <ns:procedure xlink:href="MY-SML-Described-SYSTEM"/>
        <ns:observedProperty

xlink:href="urn:ogc:def:property:OGC::dynamicLocation"/>
        <ns:featureOfInterest>
          <ns1:SamplingPoint>
            <ns1:sampledFeature xlink:href="urn:ogc:def:nil:OGC:unknown"/>
            <ns1:position>
              <gml:Point>
                <gml:pos srsName="urn:ogc:def:crs:EPSG:7.4.1:4979">
                  34.739429354667664 -86.72852575778961 239.0
                </gml:pos>
              </gml:Point>
            </ns1:position>
          </ns1:SamplingPoint>
        </ns:featureOfInterest>
        <ns:result>
          <ns2:DataRecord definition=
                              "urn:ogc:def:property:OGC::dynamicLocation">
            […]
          </ns2:DataRecord>
        </ns:result>
      </ns:Observation>
    </target>
  </EventEventRelationship>
</member>

<procedure>
  <eml:EML
    xmlns:eml="http://www.opengis.net/eml/0.0.1"
    xmlns:fes="http://www.opengis.net/fes/2.0"
    xmlns:gml32="http://www.opengis.net/gml/3.2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <eml:SimplePatterns>
      <eml:SimplePattern inputName="input" patternID="withinPattern">
        […]
      </eml:SimplePattern>
      <eml:SimplePattern inputName="input" patternID="outsidePattern">
        […]
      </eml:SimplePattern>
    </eml:SimplePatterns>
    <eml:ComplexPatterns>
      <eml:ComplexPattern patternID="entrancePattern">
        […]
      </eml:ComplexPattern>
    </eml:ComplexPatterns>
    <eml:TimerPatterns/>
    <eml:RepetitivePatterns/>
```

```
    </eml:EML>
  </procedure>
</DerivedEvent>
```

The output is encoded using the Event Model developed in OWS-6 (see OGC 09-032, section 6.5) and enhanced in OWS-7 (see OGC 10-060 section 12.2.4). The output is a derived event as its result value was derived from other events (the input) and does not only contain the value from one input event.

The main elements of the derived event are the event time, the named values, the members and the procedure. The event time is the time when this event was derived (or the described situation was detected). In this case this is short after when the second event was received which was the first event inside of the area of interest. The only (named) value is the predefined value stating that the AOI was entered.

The member events are connected to the main event by a specified relation. In this case the relation is defined as "cause" meaning that the member events were the cause for this event. The two causing input events are included to allow notified clients to extract additional information. This is necessary to correctly instruct the change detection Web Processing Service. In the Tracking and Notification service these causal ancestors are only included if the "createCausality" attribute is set to "true" in the EML event patterns.

At last the event contains the procedure which was used to derive it from the input events. This is usually the set of instructions given within the subscription. Here it is the EML document with the event patterns described in section 7.2.3.1.

7.2.4 Summary

The main advantages of using the Sensor Event Service (SES) as basis of the Tracking and Notification service implementation is the integrated use of existing standards for various purposes and the flexibility that comes along with it. Also the integration of sophisticated event processing techniques like Complex Event Processing (CEP) and Event Stream Processing (ESP) contribute large added value (OGC 10-060, chapter 9 describes some examples).

In the area of communication protocols the implemented standards are mainly SOAP from W3C and the Web Services Notification suite from OASIS. These standards provide a common way to implement the publish-subscribe messaging pattern which is the basis of the Tracking and Notification service communication. By reusing existing standards instead of defining isolated solutions interoperability with external components can be achieved far easier. For instance many Enterprise Service Bus (ESB) implementations provide a WS-Notification adapter out of the box which can be used to connect the Tracking and Notification service with a broad range of information producers and clients.

For the definition of filtering instructions the SES makes use of the OGC Filter Encoding specification (FES). It allows to define filter statements based on spatial, temporal and comparison operators and to join these via logical associations. The FES is also used for

filter statements that are embedded within the Event Pattern Markup Language (EML). It is used to encode event processing instructions. Using the EML it is possible to execute CEP and ESP based functions on the stream of incoming events and go far beyond the capabilities of simple filtering. One example is shown in the OWS-7 SFE scenario where a simple filter would only be able to detect events (camera positions) that are close to a given track. If there is one position event per second sent to the Tracking and Notification service a client would also get one notification per second if only simple filtering is used. The event processing techniques implemented allow creating instructions that detect if a position outside of the area of interest is followed by a position inside of the area and derive the information that the area was entered. This only happens once unless the area of interest is left again.

The main disadvantage that comes with the use of existing standards is a bit loss of simplicity. The specification of the SES alone is not complete as other specifications have to be consulted in order to have all necessary information. This problem however applies only at the beginning, once being familiar with the specifications one can work with them in the same way as if everything would be specified in only one document.

8 Standards and specifications relevant for and related to Dynamic Sensor Notification

This section covers a discussion of the standards and specifications related to Dynamic Sensor Notification. It is similar to the discussion included in the OWS-7 Event Architecture Engineering Report (OGC 10-060) which deals with common eventing topics. This section focuses on sensors and sensor notification.

8.1 Relevant standards and specifications

The service standards and specifications most relevant for dynamic sensor notifications are the Sensor Alert Service (SAS), the Web Notification Service (WNS) and the Sensor Event Service (SES). All three specifications are available as OGC documents, however none is released as official OGC standard yet. The most recent SAS and WNS specifications are Best Practices, the SES is available as a Discussion Paper. Another set of SWE standards that influenced and were influenced by the notification and eventing discussions are the Sensor Planning Service (SPS) and the SWE Service Model (SWES) specifications.

On the side of (meta-) data encodings the most relevant standards are Observations and Measurements (O&M), SWE Common and the Common Alerting Protocol (CAP).

8.1.1 Timeline

The following figures give an overview of the timeline of the relevant standards and specifications. The first figure (Figure 11) shows the timeline of the identified service specifications (SAS, SES, WNS and SPS) as well as the Event Service and the Event Architecture. The latter are topic of the OWS-7 Event Architecture Engineering Report

(10-060) which describes them and their relationships to the mentioned specifications in more detail.

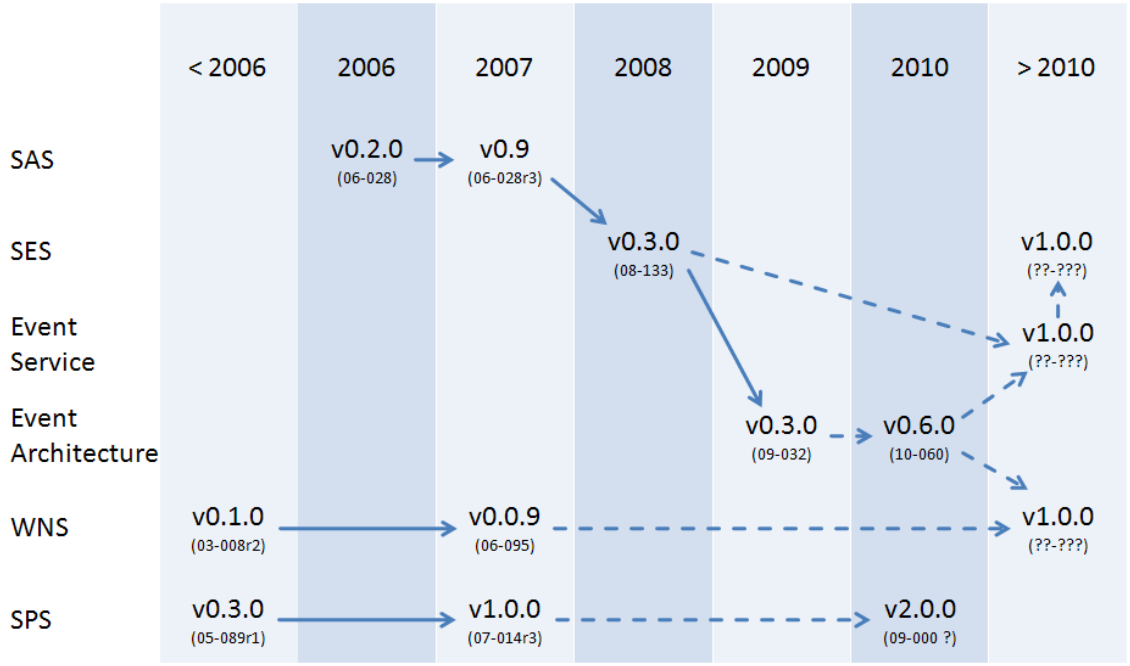

Figure 11 - Timeline of the relevant service specifications

The second figure (Figure 12) shows the timeline of the relevant encodings. As one can easily see they do not interact very much. Also for O&M, SWE Common and CAP new version are in preparation.

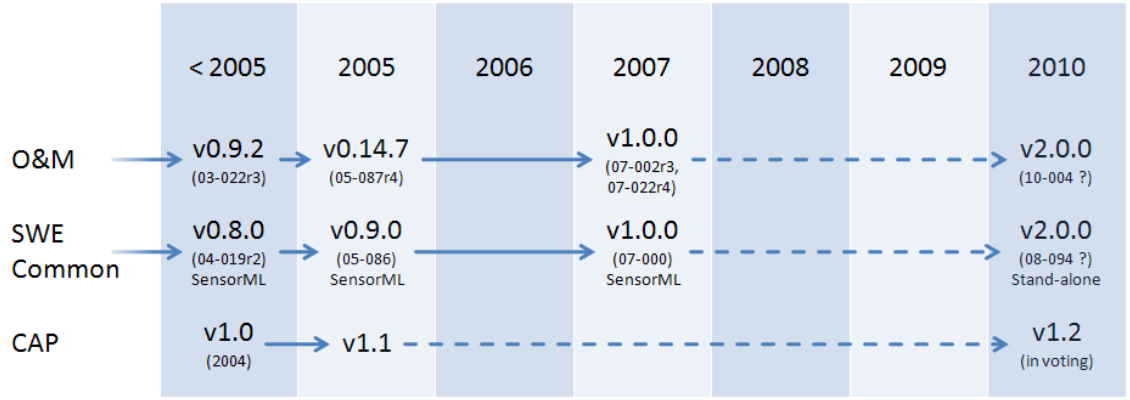

Figure 12 - Timeline of the relevant encoding specifications

8.1.2 SAS, SES and WNS

The Sensor Alert Service was designed to allow live access to sensor data (streams). A user can subscribe for sensor measurements using certain criteria like a bounding box or a value threshold. The SAS was not released as an official standard. Instead in 2008 the Sensor Event Service was designed based on the experience gained on the SAS work. It differs in a variety of points from the SAS but the central use is the same. Because it is

not released as a standard or best practice paper it can be seen as an experimental successor of the SAS.

The following table (Table 1) shows the main enhancements of the SES in contrast to the SAS. This includes more sophisticated spatial, temporal and comparison filter capabilities. This is achieved by using the OGC filter encoding as a possible encoding of subscription filter criteria. Also explicit aggregations of multiple filters using logical operator are possible.

Table 1 - Enhancements by the SES specification

	SAS	SES
Filtering:		
- spatial	(✓)	✓
- temporal	×	✓
- comparing	(✓)	✓
- aggregation	×	✓
Event Processing	×	✓
Topics	×	✓
Unite Conversion	×	✓
O&M Support	×	✓

In addition the SES allows the service to perform Event Processing. The event patterns needed as instructions for this functionality are encoded using the Event Pattern Markup Language (EML) which is available as an OGC discussion paper (08-132). More information on Event Processing can be found in the EML discussion paper and the OWS-6 SWE Event Architecture Engineering Report (09-032).

The support of topics is also introduced by the SES and allows a service provider to set up specific event channels for a subset of the published events. This way one can separate sensor measurement events from administrative events. People only interested in the latter can subscribe to the according topic instead of creating their own subscription. Thus, topics can help to save resources on filtering computations.

The SES also provides an automatic unit conversion. Measurements in a specific unit can be matched against subscriptions that use another but compatible unit. In the case of temperature measurements for instance, the SES can compare measurements using degree Fahrenheit with filter criteria using degree Celsius.

These improvements of the service capabilities led to a more heavy service and protocol. The SAS for instance sends notifications in a very short text based format. Each notification consists of just one line which contains the three dimensional position of the measurement, the measurement time and a time stamp until when the measurement is valid as well as the measured value. This encoding is very short and cheap to transport but essential information is missing which is needed for sophisticated filter functionality. For instance there is no reference to the sensor or sensor description and no indications of the measured phenomenon or the used unit of measurement.

The SES uses O&M as default encoding for notifications where all the necessary information can be encoded. This allows to integrate missing information and metadata but also to easily extend the notifications. Extensions to the SAS message format would break the compatibility; extensions to O&M encoded messages do not. Thus, it is for instance possible to include uncertainty information encoded in UncertML.

Also the SES switched from XMPP as communication protocol to HTTP and OASIS Web Services Notification (WS-N). The reason for this switch is that it was preferred to use an existing publish subscribe interface instead of defining an own. To the time of the SES development WS-N offered the most functionality and was chosen. Currently there is only a SOAP binding available for Web Services Addressing (WS-A) on which WS-N relies. More information on WS-N can be found in the OWS-7 Event Architecture Engineering Report (OGC 10-060).

The Web Notification Service is intended to deliver messages through various channels like different protocols for instance HTTP, XMPP, SMS or E-Mail. In this sense the WNS can be seen as a protocol transducer.

Also the transformation between different encodings of notifications can become capability of the WNS. This can for instance be done by applying XSLT templates on incoming notifications to generate human readable outputs.

As shown in Figure 11 the Web Notification Service also has not reached the version 1.0 and is not an official OGC standard. But unlike to the SAS/SES there was no further specification work inside the OGC.

8.1.3 O&M, SWE Common and CAP

8.1.3.1 Observations & Measurements

O&M is an application profile of GML. It is currently available as an OGC standard in version 1.0 though version 2.0 is under development. More information on O&M and its relevance for the Event Architecture can be found in the OWS-7 Event Architecture ER (OGC 10-060 section 12.2.5).

8.1.3.2 SWE Common

SWE Common defines common data types for the use in SWE. More information on SWE Common and its relevance for the Event Architecture can be found in the OWS-7 Event Architecture ER (OGC 10-060 section 12.2.6).

8.1.3.3 Common Alerting Protocol

The Common Alerting Protocol (CAP) is an OASIS standard and commonly used format for the exchange and publication of alerts. It is readable for humans and relatively simple to use. Its scope lies on the communication of alert messages. Thus, it lacks in encoding non-alert notifications like simple position information. CAP is currently available in version 1.1 though version 1.2 is under development. The possible use of CAP for

eventing is discussed in more detail in the OWS-7 Event Architecture Engineering Report (OGC 10-060 section 12.2.9).

8.1.4 SPS and SWES

Besides the SAS, the SES and the WNS also the Sensor Planning Service makes use of notifications. But the role of the SPS is different: in contrast to the others that serve as a notification broker the SPS is a notification producer. It produces new notifications for instance to inform users about task states.

The notification handling use by the SPS is defined in the SWE Service Model (SWES) specification. This specification has a similar role as the OWS Common specification. It defines common operations and metadata for SWE web services. With respect to eventing it contains notification metadata definitions for service capabilities and a chapter on publish-subscribe communication. This work focused on using WS-Notification in SWE eventing contexts.

The development of the SWES specification was influenced by the SPS and the SOS but also the Event Architecture and thus indirectly the SES as well (see Figure 13).

The SPS and SWES standard working groups are currently reviewing the comments received in the RFC phase and - once all comments have been processed - will move forward to vote for adoption as OGC Standards.

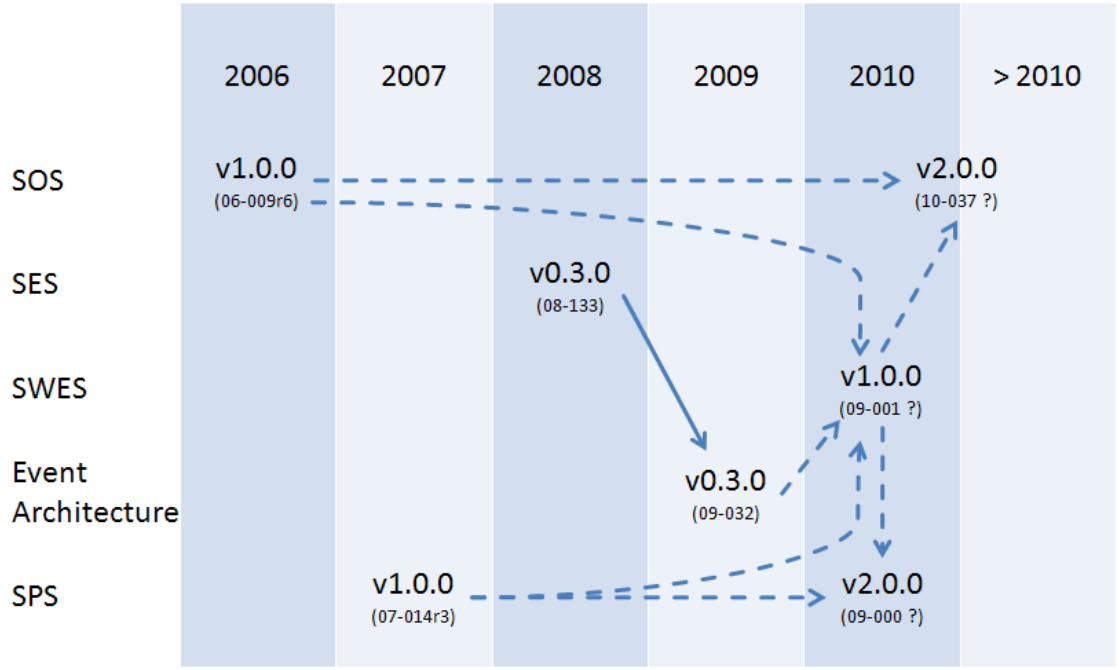

Figure 13 - Relations of the SWE Service Model specification to other specifications

8.2 Future Prospects

The major lack in the area of eventing in the OGC and thus for dynamic sensor tracking is the nonexistence of an official OGC standard. As described above the SAS, SES and WNS are not in this state. The plan to get there is indicated in Figure 11 and described in the OWS-7 Event Architecture ER (OGC 10-060). In parallel to the practical experience with the SAS and SES specifications, abstract publish / subscribe functionality is developed. The results shall be fed into a common Event Service specification which shall be applicable in all areas of the OGC and eventually released as official standard.

Once this service specification is available it has to be evaluated with respect to sensor specific requirements. This may for instance be the definition of a SWE set of event channels for measurements, sensor status information and so on. Also the use of the "eventSourceMetadata" should be defined for instance to be encoded in SensorML.

Such restrictions to the common Event Service specification should be gathered in a single document and published as a best practices document. This would then together with the Event Service standard take the role of the Sensor Event Service specification replacing the current SAS and SES specifications.

Another possible way would be to define OGC Event Service compatible extensions for the SWE specifications like SWES, SPS and SOS. This could simplify the management of the specific definitions for events, event encodings, event metadata and event channels. Which way to take should be discussed when the OGC Event Service specification is available.

At this time, also the Web Notification specification should be updated to be compatible with the OGC Event Service. This would allow the delivery of notifications out of the SWE infrastructure to multiple different targets like decision makers or external warning systems.

Bibliography

[1] W3C, Efficient XML Interchange Working Group, online at http://www.w3.org/XML/EXI/

[2] W3C, Efficient XML Interchange Evaluation, online at http://www.w3.org/TR/exi-evaluation/

