Bigagli and Rieke *Open Geospatial Data, Software and Standards* (2017) 2:18
DOI 10.1186/s40965-017-0030-7

Open Geospatial Data, Software and Standards

ORIGINAL ARTICLE Open Access

The new OGC Publish/Subscribe Standard - applications in the Sensor Web and the Aviation domain

Lorenzo Bigagli[1*] and Matthes Rieke[2]

Abstract

The Open Geospatial Consortium (OGC) has conducted much work in the past on event-based models and architectures. However, the current OGC standard baseline only supports synchronous web service capabilities, which have insofar primarily addressed the request/reply model, where a client makes a request and the server usually responds synchronously, with either the requested information or a failure. Recently, the OGC Publish/Subscribe 1.0 Standard has introduced an abstract model for publish/subscribe message exchange, a long-awaited building block in the OGC suite of geospatial standards. The publish/subscribe pattern is distinguished from the request/reply one by the asynchronous delivery of messages and the ability for a client to specify an ongoing, persistent expression of interest. In this work, we report on the experimentation of the new OGC Publish/Subscribe 1.0 Standard in the context of the OGC Testbed-12 initiative and related fields of work, particularly in the application domains of Sensor Web and Aviation. We illustrate and discuss the enhancements in comparison to previous OGC service architectures, highlighting the benefits of introducing the PubSub 1.0 Standard into the considered systems and their workflows.

Background

OGC standards have primarily addressed the request/reply model, as it is sufficient to meet many use cases. Clients request data of interest when required and may request updates later on. In the request/reply model, a client makes a request and the server usually responds synchronously, with either the requested information or an error report. This provides relatively immediate feedback, but may be impractical in application domains characterized by processes with long runtime, such as model execution in the Model Web, or distributed search in geospatial discovery, where clients need to keep the connection to the server continuously open, in order to wait for the responses.

Besides, synchronous communication may be insufficient when the application depends on asynchronous events, such as external communications, status changes, natural phenomena, or data updates. In such cases, polling strategies are usually implemented where clients repeatedly check for the desired information. This has undesirable side effects: if a client polls too frequently, server load and network traffic might be increased; if a client polls too infrequently, it may not be notified when needed. These issues are aggravated when event occurrences are unpredictable, or when the delay between event occurrence and client notification must be kept small.

The recently approved Open Geospatial Consortium (OGC) Publish/Subscribe 1.0 (in short, PubSub) Standard defines a mechanism to support publish/subscribe requirements across OGC service interfaces and data types, including coverages, features, and observations. The PubSub Standard consists currently of two parts: a Core document [3] which provides an abstract description of the mandatory functionality, independent of the underlying binding technology, along with several extension classes for optional capabilities; and a SOAP binding document [4], which defines how the PubSub functionality is realized in SOAP services, i.e. leveraging the OASIS Web Services Notification (WS-N) set of standards [7]. Work is underway to define a RESTful binding, which will specify how to realize the PubSub functionality in RESTful services.

* Correspondence: lorenzo.bigagli@cnr.it
[1]Institute of Atmospheric Pollution Research, National Research Council of Italy, Florence, Italy
Full list of author information is available at the end of the article

PubSub is a long-awaited building block in the OGC suite of geospatial standards, and a fundamental enabler towards an event-driven Service-Oriented Architecture (SOA). Its roots can be traced back to 2006, when an OGC Sensor Alert Service was proposed, reappearing through the years as a Sensor Event Service (SES) and several evolutions of an Event Architecture. In 2010, the PubSub Standard Working Group was officially chartered, but it was not until 2015 that the 1.0 draft was finalized and eventually officially approved in February 2016. Even before the official documents were published, work was already ongoing on several extensions that would introduce PubSub capabilities to well-known OGC Web Service (OWS) interfaces, such as Sensor Observation Service (SOS), Web Coverage Service (WCS), Web Feature Service (WFS) and Catalogue Service for the Web (CSW).

In general, the implementation of forms of asynchronous client-server interaction is highly relevant to most Spatial Data Infrastructure application scenarios. In this work, we focus on the Sensor Web and the Aviation application domains, to exemplify the benefits of the publish/subscribe Message Exchange Pattern (MEP).

In the next chapter, we introduce the concepts underpinning the PubSub Standard and some of its related technologies. Then we describe the methodological framework of our experimentation. Subsequently we describe our application of PubSub to the geospatial domains of Sensor Web and Aviation. Finally, we discuss our results, draw some conclusions and propose further activities for future research and experimentation.

Concepts and technologies

Two primary actors characterize the publish/subscribe MEP: a Publisher, which is publishing information; and a Subscriber, which expresses an interest in all or part of the published information, by subscribing to the Publisher. After a subscription has been created, the Publisher delivers information that match the subscription criteria to the Receiver defined in the subscription. Information may be XML, binary data, or other content, and is contained and transported in messages. Messages may include additional data, including headers or other information used for routing or security purposes.

In many cases, the Subscriber coincides with the entity to which messages are to be delivered (the Receiver). However, they are distinguished in PubSub to allow for these roles to be segregated. This would allow, for example, a system manager to act as a Subscriber and set up information flows from Publishers to a number of system entities that act as Receivers.

Similarly, the Publisher often coincides with the entity which delivers the data (the Sender), but these roles can be decoupled. Senders may be unaware of the ultimate recipients of their messages and of the architecture of the system into which they inject messages.

Figure 1 shows the operations required to implement the basic publish/subscribe workflow. The Subscriber creates a subscription on behalf of a Receiver using the *Subscribe* operation on a Publisher (1.0). If the Publisher accepts the *Subscribe* request, it creates a Subscription (1.1) and returns a SubscribeResponse – either success or an exception (1.2). The Subscriber may supply filter criteria with the *Subscribe* request. Filter expressions evaluate to a boolean value when applied to a message. Filter criteria can apply to the message content (such as XPath or OGC Filter Specification), the message metadata (such as header content), or to other aspects of a message.

Whenever a new message is available, the Publisher attempts to match it against each Subscription (2.0). Those messages that evaluate to true for all filter expressions on a Subscription are considered to have matched that Subscription. If the message matches a Subscription, the Sender delivers it to the location and/or Receiver specified for the Subscription via the *Notify* operation (2.1). Messages are delivered asynchronously as they become available to the Publisher.

Every Subscription has a defined time at which it expires. When that time is reached, the Publisher terminates the Subscription. The *Renew* operation may be utilized (3.0) to set a new termination time for a Subscription. If the Publisher accepts the *Renew* request, the new termination time is set on the Subscription and the Publisher returns a RenewResponse (3.1) informing the Subscriber of the outcome of the request.

Termination of a Subscription may be requested any time after the creation via the *Unsubscribe* operation (4.0). If the Publisher accepts the *Unsubscribe* request, it terminates the subscription (4.1) and returns an UnsubscribeResponse (4.2) informing the Subscriber of the outcome of the request.

The PubSub Core document attributes the above basic functionalities to the Basic Publisher conformance class. PubSub Core defines several other optional conformance classes (see Fig. 2), introducing additional functionalities, e.g. to pause a Subscription (Pausable Publisher), derive additional publications (Publication Manager), group messages in batches (Message Batching Publisher).

PubSub Core requires that a PubSub-enabled OWS advertise the implemented Conformance Classes in its Capabilities document, namely in the Profile property of the ServiceIdentification section. Besides, it requires that a Publisher return the additional Capabilities components represented in Fig. 3 in its GetCapabilities response. The PubSub Standard does not specify the specific mechanism for incorporating these additional Capabilities components into an OWS Capabilities document. We have proposed to include these additional

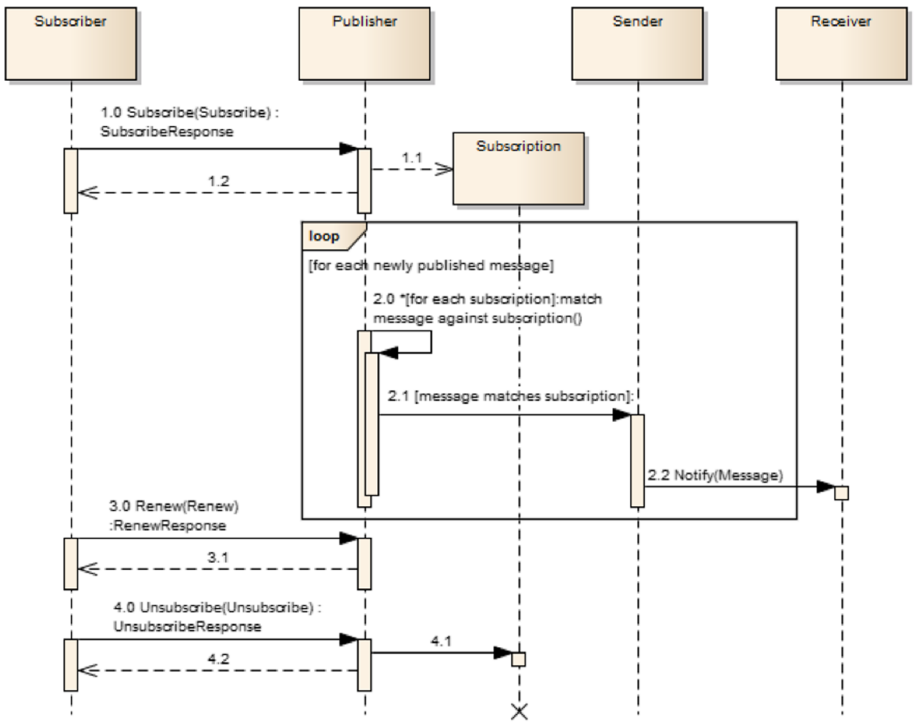

Fig. 1 Basic publish/subscribe workflow (source: [3])

Capabilities components in the ExtendedCapabilities of the OWS, as detailed in the following.

The FilterCapabilities component describes the filtering-related capabilities of a PubSub-enabled OWS, i.e. the filter languages it supports for matching messages against subscriptions (e.g., OGC Filter Encoding, XQuery).

The DeliveryCapabilities component describes the methods supported by the PubSub-enabled OWS for delivering messages, e.g. SOAP, WS-N, ATOM, Server-Sent Events (SSE), WebSockets, Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH).

PubSub is agnostic as for the language to filter messages in subscriptions and as for the delivery of messages. Publishers may support multiple filter languages, and offer more than one method of delivery for each Publication, to be chosen by each Subscriber. This supports the flexible pluggability of technologies. The publish/subscribe MEP typically implies push-style message

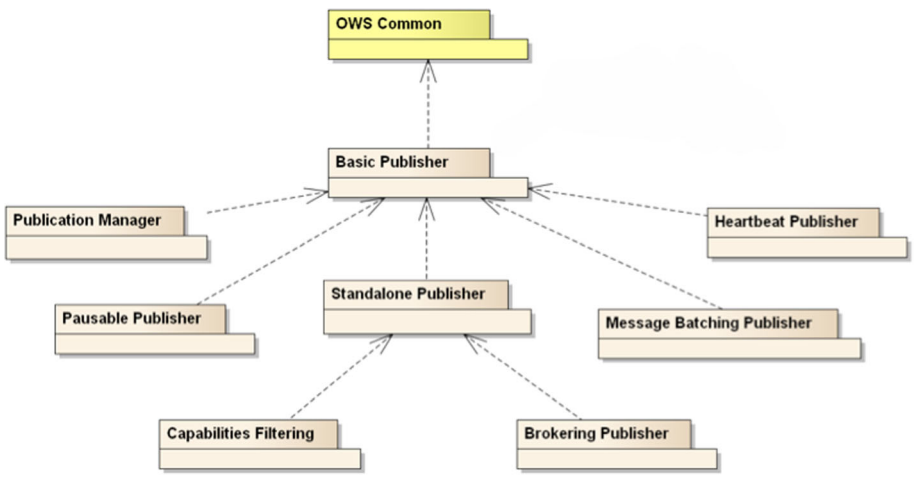

Fig. 2 PubSub Core Conformance Classes (source: [3])

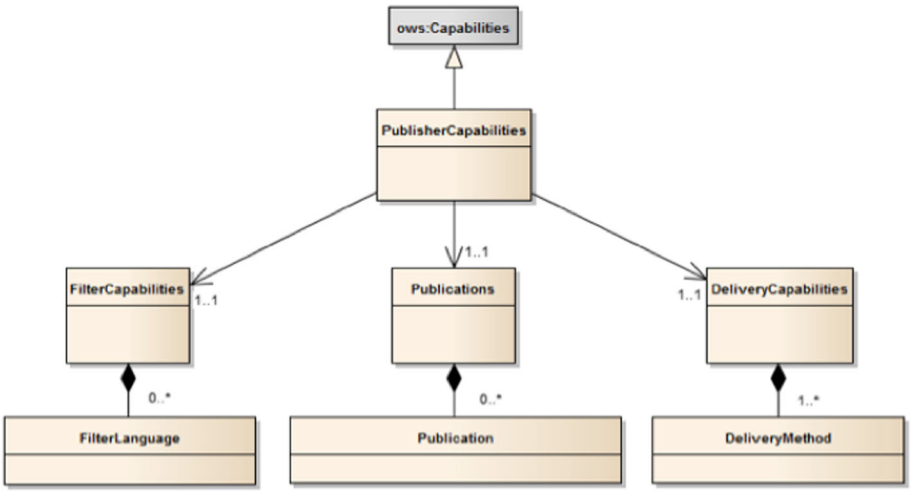

Fig. 3 PubSub Capabilities components (source: [3])

delivery, however some delivery methods may actually be underpinned by pull-based mechanisms (e.g. polling).

Examples of delivery methods include: SOAP and related technologies, such as WS-N (used by the PubSub SOAP Binding), ATOM, PubSubHubbub, OAI-PMH, email, Short Message Service, WebSockets and SSE. SSE is a pure push-style communication technology based on HTTP and the SSE EventSource API standardized as part of HTML5 by the W3C. An SSE client (e.g. all modern HTML 5.0 browsers) receives automatic updates from a server via HTTP connection by setting some HTTP parameters in the request.

The Publications component describes the contents offered by the PubSub-enabled OWS, i.e. the sets of messages that Subscribers can subscribe to. PubSub is agnostic as for the message encoding. The most generic mechanism to notify about updates is that the Publisher re-send the whole response element corresponding to the request used as filter in the Subscription. For example, in the case of WFS, if the client subscribes with a wfs:GetFeature request as a filter, the PubSub-WFS should notify about any changes by delivering a standard wfs:FeatureCollection, in response to that request. By receiving the new response and comparing it with the previous one, a Subscriber can figure out the changes. Future evolutions of this extension may evaluate more efficient and semantically accurate encoding of notifications. A possible option for XML-based content types is XMLdiff (e.g. XML Patch, RFC 5261), or annotations (XML attributes) to add simple CRUD semantics on top of the existing XSDs.

Methods

Major parts of the experimentation reported in this work were conducted within the 12th edition of the OGC Testbed initiative. The OGC is an international not for profit organization, whose members (about 520 at present, from government, commercial organizations, NGOs, academic and research organizations) are committed to making quality open standards for the global geospatial community. These standards are made through a consensus process and are freely available for anyone to use, to improve sharing of the world's geospatial data. OGC standards are used in a wide variety of domains including Aviation, Environment, Defense, Health, Agriculture, Meteorology, Sustainable Development and many more.

OGC Testbed initiatives are fast-paced, multi-stakeholder collaborative efforts to define, design, develop, and test candidate interface and encoding specifications; to experiment with new architectural approaches and patterns; to develop guides on spatio-temporal information technologies; and to serve as rapid prototyping environments for general geospatial IT challenges. They are executed as part of the OGC Innovation Program (IP), but receive requirements and definition of work items from all OGC programs (i.e. Standards Program, Compliance Program and Communications & Outreach Program). The results of a Testbed are captured in Engineering Reports (ERs) and demonstrated with prototype implementations.

OGC Testbed-12 started in January 2016 and finished with the final demonstration event in November 2016. Several work items addressed the means to incorporate forms of asynchronous service interaction, including the publish/subscribe MEP, for example in Web Processing Service (WPS), WCS, WFS, or in geospatial queries of aviation data. The experimentation reported in this work contributed mainly to the Large-Scale Analytics (LSA) and Aviation threads of Testbed-12. Details on these are documented in the Testbed-12 Results Engineering Report [11].

In particular, the LSA thread included the Asynchronous Service Interaction subtask, part of a set of subtasks aimed at enhancing the OGC Baseline, by extending OGC architectural designs through efforts that cross over several individual standards and services and are applied in a much wider scope. The activities of the subtask have experimented and compared three different approaches to implement asynchronous interaction in OGC Web services:

1. WPS façades;
2. Specific extensions to each OGC Web Service with asynchronous request/reply capabilities;
3. PubSub Standard.

The results of these activities are captured in the document deliverable "A067 Implementing Asynchronous Service Response Engineering Report" [8], which focused on the first and the second approach, with the goal to summarize and compare the results from using a WPS façade and an extension for WFS for asynchronous service responses, as well as to provide recommendations for future activities. The document deliverable "A074 PubSub/Catalog Engineering Report" [1] focused on the third approach, OGC PubSub, in the specific case of catalogs, investigating the functional requirements of an interoperable, push-based data discovery solution.

The Aviation thread addressed the requirements for Asynchronous Messaging for Geospatial Queries of Aviation Data. This task investigated the means to incorporate publish-subscribe messaging patterns for the retrieval of aviation data (i.e. AIXM, FIXM, or AMXM) information using geospatial queries through a Java Message Service (JMS) interface and Advanced Message Queuing Protocol (AMQP) interface. This task also demonstrated a capability using the recommended approach.

The results of these activities are captured in the document deliverable "Asynchronous Messaging for Aviation Engineering Report" [9] describing the results of the asynchronous messaging study and implementation applying the PubSub Standard in the aviation domain. The goal was to develop a publish/subscribe messaging architecture between different aviation components such as clients, data provider instances, and Data Brokers. The architecture study took various messaging protocols such as AMQP 1.0, JMS and OASIS WS-N into consideration. Special attention was laid on the smooth integration of middleware solutions implementing the various messaging protocols and the PubSub Standard.

Application domains

The dissemination of near-real time data plays an important role in many application domains. This section provides examples where the OGC PubSub Standard has been integrated into existing architectures in order to provide an interoperable solution for the asynchronous delivery of data.

Sensor web

Sensor Web technologies have been driven by the OGC Sensor Web Enablement (SWE) Working Group since more than a decade. The SOS is a well-established interface standard for providing access to timeseries observation data. As it follows the request/reply MEP, the SOS does not provide access to near-real time data. Still, many application domains of the Sensor Web require access to data in an asynchronous way.

Water infrastructure management highly depends on low latency data processing. In particular, the exceedance of critical thresholds of water gauges play an important role in the prevention of flood scenarios. In the past, error prone polling mechanisms using SOS instances have been installed to achieve this goal.

Another example of a Sensor Web application domain is the management of air quality data. Near real-time processing of air quality data has become an increasingly important mechanism to detect critical air pollution situations (e.g. for large cities, see [2]). Sensor Web technologies and in particular PubSub provide the means to deliver such data in time using asynchronous message delivery. This section describes the approach and design of a Sensor Web service architecture that addresses these requirements.

Technological viewpoint

In order to achieve (near) real-time dissemination of observation data, previous Sensor Web service infrastructures have been based on the orchestration of multiple server nodes and intermediary components. The subscription to as well as the delivery of data of interest was realized by either a dedicated service component or a customized workaround. Bröring et al. [5] describe a complex architecture to achieve this functionality featuring OGC service specifications such as SES and Web Notification Service (WNS). In analogy, Fig. 4 illustrates the design of such an architecture.

The architecture is comprised of four service nodes: a *SOS* that provides access to time series observation data, a *SES* that provides the possibility to subscribe for specific data and an intermediary component, the *SOS-SES Connector*. In addition, the *WNS* has been used to establish the communication between SES and the client. The dissemination mechanism for real-time observation data could be email delivery, HTTP for machine-to-machine communication or for example an instant messaging protocol such as XMPP (Extensible Messaging and Presence Protocol).

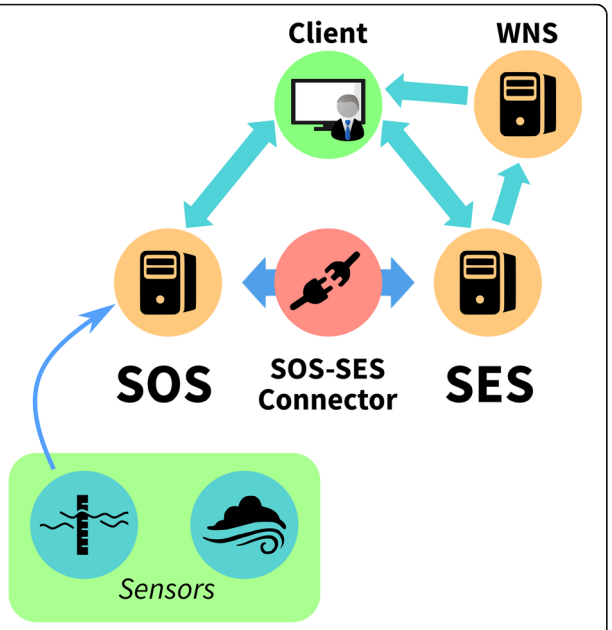

Fig. 4 Legacy Sensor Web Architecture

The SOS-SES Connector component introduced a fair amount of instability to the system: to manage the retrieval of time series observation data, the component had to persist its state. In particular, it was required to store the timestamps of single observations in order to not retrieve these multiple times and subsequently produce duplicates. The introduction of such business logic made the system error prone, e.g. in situations where the connector's state could not be persisted and restored after a planned server maintenance. In general, the involvement of such a high amount of service instances implies multiple drawbacks:

1. More web service components have to be maintained, configured and updated
2. every component introduces a possible point of failure
3. the overall system becomes static and prevents extensibility
4. for each service endpoint, a corresponding client and/or management component has to be provided

Real world applications of this architecture pattern have confirmed the above listed aspects. Therefore, the development of the OGC PubSub specification has been highly anticipated in the Sensor Web community as it provides the capabilities to overcome multiple of these concerns. Figure 5 illustrates a Sensor Web architecture that introduces PubSub as a pivotal component.

The SOS service is extended with the PubSub interface in a modular way. All functionality that has previously been realized by SES, the SOS-SES Connector and the WNS is now bundled within the PubSub interface. The client software only communicates with one service instance that provides both SOS and PubSub interfaces.

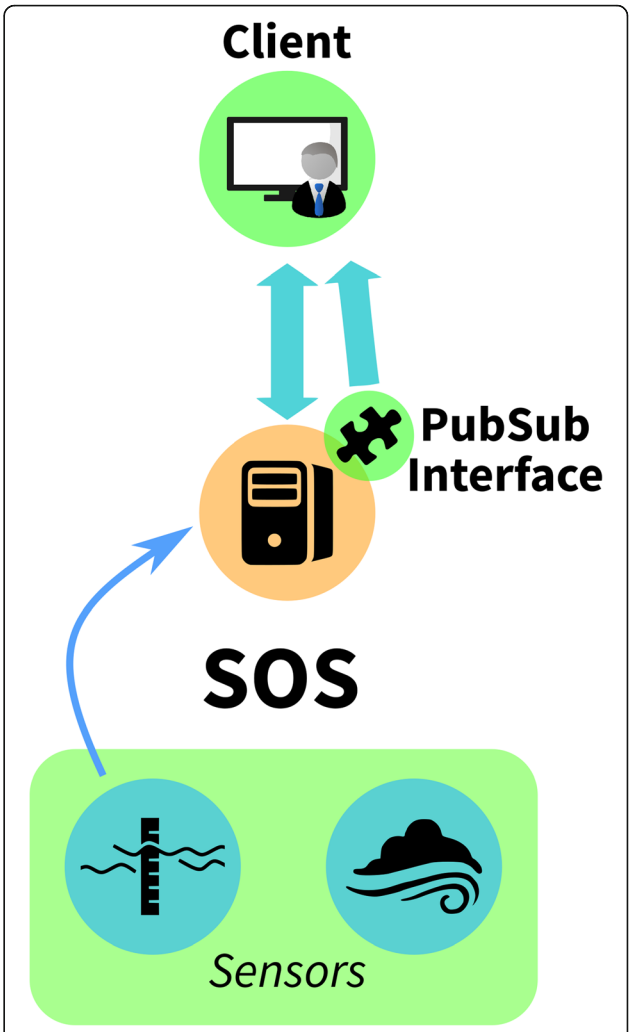

Fig. 5 PubSub-enabled Sensor Web Architecture

A PubSub service instance has to provide a set of *Publications* (see section "Concepts and technologies") that form the basis for a subscription. In the Sensor Web domain, it has become a common practice to provide the *Observation Offerings* of the SOS as the *Publications* offered by the PubSub interface. *Observation Offerings* bundle the time series data for a given context that is valid for the use case. For example, in a water infrastructure architecture, all measurements of a certain water gauge station are defined as one *Observation Offering*. A typical client application consists of a mechanism to display time series data for distinct measurement stations (see Fig. 6), i.e. the offering.

The integration of SOS and PubSub by means of the Observation Offering shows great potential for user applications and provides a close link to an existing concept of

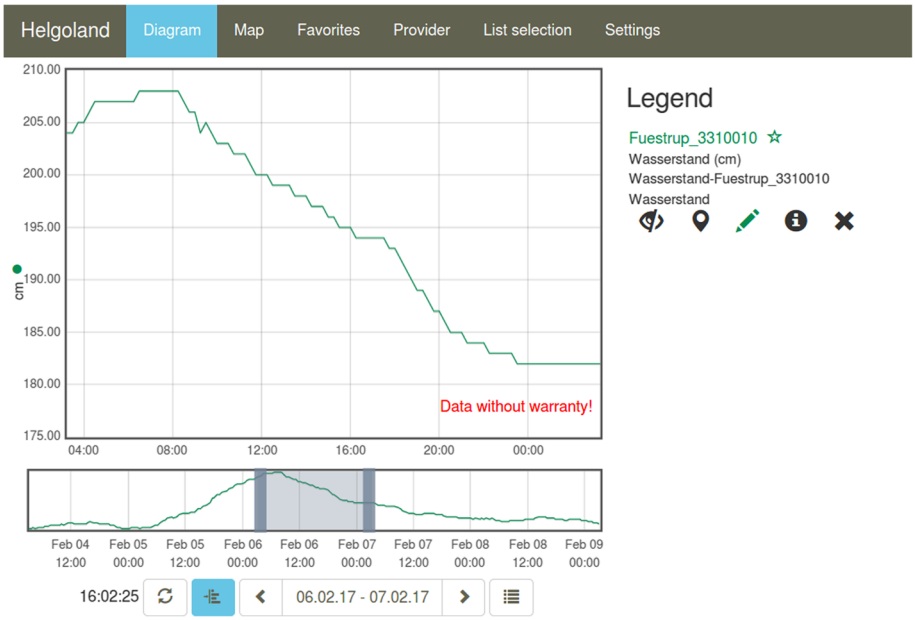

Fig. 6 Helgoland Sensor Web client

the SOS. If the user is interested in real-time data by a certain measurement station, he/she can subscribe for the publication that is the same as the offering. An integration within the user interface could be realized via a few controls within the diagram view, e.g. via a button "auto-update" to enable the real-time update of the diagram with new observation data. For such a web-based solution, the Delivery method would be pre-defined to use WebSockets. The web application would act as both the Subscriber and Receiver following the PubSub terminology.

Aviation and air traffic management

This section describes previous and current service architectures in the domain of Air Traffic Management (ATM). Current ongoing initiatives such as the SESAR Joint Undertaking governed by Eurocontrol [10] and the System Wide Information Management (SWIM) NextGen of the Federal Aviation Administration (FAA) project [6] investigate the possibilities to integrate the asynchronous delivery of ATM data. In this scope, several architectures have been developed (mainly within OGC interoperability experiments) to design a possible SOA featuring publish/subscribe capabilities.

Use cases

Base data such as airspaces or scheduled flights are modeled and encoded using the Air Traffic Information Exchange Model (AIXM) and related technologies (e.g. Flight Information Exchange Model, see [9]). Access to this base data is provided via WFS instances. ATM requires to communicate updates on specific features in a fast and reliable way. For example, the runway of an airport could be closed due to bad weather, or an ad-hoc airspace could be defined to ensure the secure deployment of Search and Rescue activities. This information is highly dynamic and therefore requires an approach beyond the request/reply MEP as provided by classic data services like the OGC WFS.

Technological viewpoint

The OGC has driven the development of SOAs in the Aviation and ATM domain since 2008. From the beginning, the push-based dissemination of data has played a fundamental role in the design of systems and its applications. Figure 7 outlines the high-level architecture for asynchronous data delivery of the OGC Web Services Testbed – Phase 9 (OWS-9). The Event Service (ES) takes the role of a middleware component, which manages subscriptions, as well as the corresponding delivery of ATM data to clients.

The WFS instance had to take care to push updated data to the ES instance which then delivered the data to subscribed client components. This architecture showed drawbacks in terms of flexibility and interoperability. In particular, both the client component and the WFS instance had to communicate with the interfaces of the ES. In addition, the ES had to communicate with the WFS in order to enrich data with supplementary information. This was especially the case for AIXM data as updates to features in general only provided the specific updated properties of the feature and not the full representation. Still, the ES might have had to apply a spatial

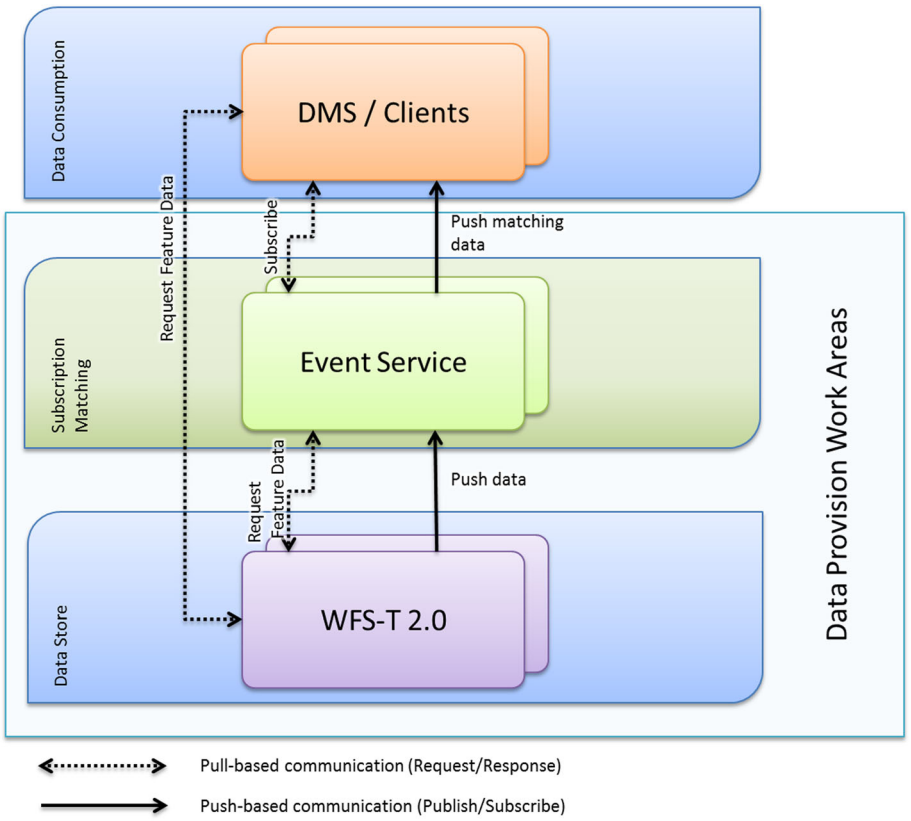

Fig. 7 Push-based architecture as designed in OWS-9 (source: [12])

filter as defined in a subscription and therefore had to resolve relevant information in an additional processing step.

Recently, the OGC designed an advanced architecture which features the PubSub Standard as part of the Testbed-12 interoperability experiment (see [9]). Figure 8 illustrates the newly developed architecture. It acts as the intermediary step towards a single-service architecture where both the request/reply access to data and the asynchronous delivery is managed in a single component.

The Asynchronous Messaging Server is an implementation of the OGC PubSub Brokering Publisher conformance class. It is therefore able to manage subscriptions as well as to act as a data broker between different components of the system architecture. AMQP in its version 1.0 has been used as the delivery method. This allowed the integration of existing data delivery components such as the Harris DEX, which is the current implementation of FAA NAS (National Airspace System) Enterprise Messaging Service.

In contrast to the application within the Sensor Web domain, the definition of PubSub publications has been realized in a broader way. As the system architecture was of a prototypical nature, it was sufficient to defined general publications, i.e. one for AIXM data and FIXM data. In production environments, one possibility for a finer-grained set of publications could be a definition based on geographic regions (e.g. "Airspaces in the San Francisco area"). This would fit to most use cases of ATM as a subscription would likely be defined in the scope of a specific flight and therefore would have a predictable spatial extent.

Results and Discussion

The publish/subscribe MEP is distinguished from the request/reply model by the asynchronous delivery of messages from the server to the client, and the ability for a Subscriber to specify an ongoing, persistent expression of interest.

In the publish/subscribe model, the server can take the initiative to notify the clients when an event occurs, rather than relying on clients to anticipate it, hence reducing the latency between event occurrence and event notification.

The two models are complementary, not alternative. In fact, request/reply may be supplemented with publish/subscribe: for example, the initial state of interest may be requested via request/reply, and then a subscription may be created so that changes and updates are

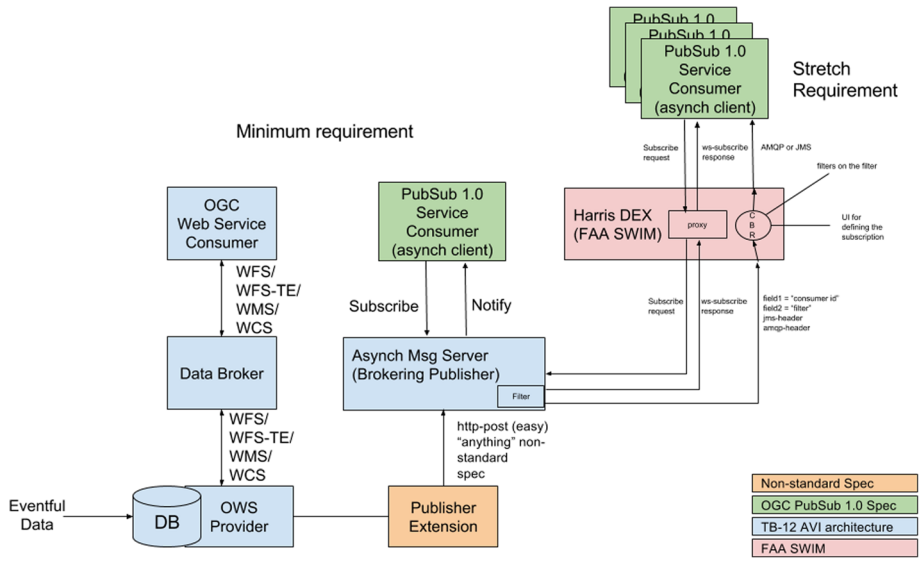

Fig. 8 PubSub architecture as designed in OGC Testbed-12 (source: [9])

delivered. This can result in less or more predictable network traffic.

In fact, the publish/subscribe MEP can be implemented for the generic OWS via its existent operations, as exemplified in Fig. 9.

Our work has provided detailed extensions, implementation experiences as well as a proposal for a general, basic mechanism for enabling to PubSub the generic OWS over the existing request/reply MEP, as documented in the PubSub / Catalog Engineering Report [1] and the Asynchronous Messaging for Aviation Engineering Report [9].

The Testbed-12 Results Engineering Report [11] discusses the three approaches considered in Testbed-12 to implement asynchronous communication: WPS façades, service-specific extensions, and OGC PubSub. The first builds on the capabilities of the WPS, which offers asynchronous communication patterns and therefore can be used as façade to any other service. The second develops an individual solution per each OWS interface (in fact, the WPS asynchronous interface is such a solution, for the WPS), and the third option builds on the PubSub Standard, which defines publish/subscribe functionality independently of the binding technology.

The Implementing Asynchronous Services Response Engineering Report [8] summarizes and compares the results from asynchronous communication experiments executed in Testbed-12. Testbed-12 implemented the WPS façade approach against WFS and WCS service instances, added support for asynchronous communication to a WFS using additional request parameters, and added publish/subscribe support to catalogs, as detailed in the PubSub/Catalog Engineering Report [1].

The Testbed-12 Results Engineering Report [11] concludes that all the three solutions have been implemented successfully. However, the solution based on the PubSub Standard resulted the most flexible of all three (and can include the other two if necessary).

Conclusion

The implementation of efficient asynchronous communication is necessary to support the transition to advanced push-based service interaction styles and enable ubiquitous sensor-based applications. The realizations of publish/subscribe architectures in the domains of Sensor Web and Air Traffic Management show high potential. In situations requiring the delivery of critical information in near-real time, a PubSub component can fill the

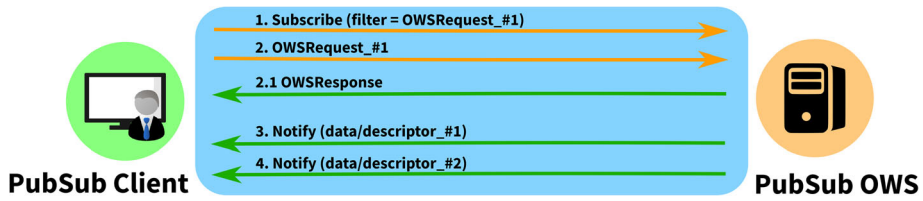

Fig. 9 Publish/subscribe MEP for the generic OWS

gap that exists when using request/reply web services. It is planned to implement OGC PubSub in additional application domains of the Sensor Web, such as oceanography and marine applications.

In conclusion, the long-awaited Publish/Subscribe 1.0 Standard bridges a gap between the existing OGC standard baseline and the many technologies supporting server-initiated communications. It is applicable to all existing OWS's by supplementing the existing service interface with additional capabilities for retrieving data in an asynchronous way.

Future work in the scope of the OGC PubSub Standard Working Group will focus on development of additional interface binding documents (e.g. REST/JSON binding) to address the needs of modern web client based systems. In addition, the definition of profiles for delivery methods is an important task. In order to allow smooth and interoperable integration of PubSub services, such profiles are crucial. The integration of modern and well-established technologies such as AMQP, MQTT and JMS into PubSub with specific profiles is therefore a central goal.

Currently, the PubSub Standard defines ways to filter data of a publication in an extensible way. An interesting field of work would be the addition of event stream processing capabilities. Stream processing goes beyond the capability of basic data filtering. It generates higher level information from raw data stream (e.g. event patterns such as "temperature < 20 followed by temperature >= 20"). It would allow the integration of concepts such as Complex Event Processing into the standard and could be realized as a dedicated event processing extension of the PubSub Standard.

Abbreviations

AIXM: Aeronautical Information Exchange Model; AMQP: Advanced Message Queuing Protocol; AMXM: Aerodrome Mapping Exchange Model; ATM: Air Traffic Management; CSW: OGC Catalogue Service for the Web; ES: Event Service; FAA: Federal Aviation Administration; FIXM: Flight Information Exchange Model; HTML: Hypertext Markup Language; JMS: Java Message Service; KVP: Key-Value Pair; LSA: Large-Scale Analytics; MEP: Message Exchange Pattern; NAS: National Airspace System; OAI-PMH: Open Archives Initiative Protocol for Metadata Harvesting; OGC: Open Geospatial Consortium; OWS: OGC Web Services; PubSub: OGC Publish/Subscribe 1.0 Standard; SES: OGC Sensor Event Service; SOS: OGC Sensor Observation Service; SSE: Server-Sent Events; SWE: Sensor Web Enablement; SWIM: System Wide Information Management; WCS: OGC Web Coverage Service; WFS: OGC Web Feature Service; WNS: OGC Web Notification Service; WPS: OGC Web Processing Service; WS-N: OASIS Web Services Notification; XML: Extensible Markup Language; XMPP: Extensible Messaging and Presence Protocol

Acknowledgements

Lorenzo Bigagli is grateful to Fabrizio Papeschi and Massimiliano Olivieri, from the National Research Council of Italy, for their contributions to the research activities leading to these results.

Funding

The research activities leading to these results were partially funded by the OGC Testbed-12 sponsors, in particular by the National Geospatial-Intelligence Agency (NGA), US Federal Aviation Administration (FAA) and European Organisation for the Safety of Air Navigation (EUROCONTROL).

Availability of data and materials

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

LB was the main contributor to the introduction, methodology, concepts and technologies chapters. MR was the main contributor to the application domains chapter. Both authors contributed to the discussion and conclusion chapters. Both authors read and approved the final manuscript.

Consent for publication

Not applicable.

Ethics approval and consent to participate

Not applicable.

Publisher's Note

Author details

[1]Institute of Atmospheric Pollution Research, National Research Council of Italy, Florence, Italy. [2]52°North Initiative for Geospatial Open Source Software GmbH, Münster, Germany.

Received: 1 March 2017 Accepted: 6 June 2017
Published online: 03 August 2017

References

1. Bigagli L. OGC® Testbed-12 PubSub / Catalog Engineering Report. 2017. Available via OGC. http://docs.opengeospatial.org/per/16-137r2.html.
2. Botts M, Percivall G, Reed C, Davidson J. OGC® Sensor Web Enablement: Overview And High Level Architecture. 2007. Available via OGC. http://portal.opengeospatial.org/files/?artifact_id=25562
3. Braeckel A, Bigagli L, Echterhoff J. OGC® Publish/Subscribe Interface Standard 1.0 – Core. 2016. Available via OGC. http://docs.opengeospatial.org/is/13-131r1/13-131r1.html
4. Braeckel A, Bigagli L. OGC® Publish/Subscribe Interface Standard 1.0 SOAP Protocol Binding Extension. 2016. Available via OGC. http://docs.opengeospatial.org/is/13-133r1/13-133r1.html
5. Bröring A, Echterhoff J, Jirka S, Simonis I, Everding T, Stasch S, Liang S, Lemmens R. New Generation Sensor Web Enablement. 2011. http://dx.doi.org/10.3390/s110302652
6. FAA SWIM. System Wide Information Management (SWIM) – FAA. 2017. https://www.faa.gov/nextgen/programs/swim/. Accessed 28 Feb 2017.
7. Graham S, Hull D, Murray B. Web Services Base Notification 1.3. 2016. Available via OASIS. http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf.
8. Proß B. OGC® Testbed-12 Implementing Asynchronous Services Response Engineering Report. Pending Approval by OGC. 2017.
9. Rieke M, Balaban A. OGC® Testbed-12 Asynchronous Messaging for Aviation Engineering Report. 2017. Available via OGC. http://docs.opengeospatial.org/per/16-017.html.
10. SESAR Joint Undertaking. SESAR – Partnering for smarter aviation. 2017. http://www.sesarju.eu/. Accessed 28 Feb 2017.
11. Simonis I. OGC® Testbed-12 Results Engineering Report. 2017. Available via OGC. http://www.opengeospatial.org/projects/initiatives/testbed12.
12. Speed C. OGC® OWS-9 Aviation Architecture Engineering Report. 2013. Available via OGC. https://portal.opengeospatial.org/files/?artifact_id=51823