Transactions in GIS

Research Article

Transactions in GIS, 2014, ••(••): ••–••

Asynchronous Geospatial Processing: An Event-Driven Push-Based Architecture for the OGC Web Processing Service

Rene Westerholt* and Bernd Resch*

**GIScience Research Group, Institute of Geography, Heidelberg University*

Abstract

Geospatial processing tasks like solar potential analyses or floodplain investigations within flood scenarios are often complex and deal with large amounts of data. If such analysis operations are performed in distributed web-based systems, technical capabilities are mostly not sufficient. Major shortcomings comprise the potentially long execution times and the vast amount of messaging overhead that arise from common poll-based approaches. To overcome these issues, an approach for an event-driven architecture for web-based geospatial processing is proposed within this article. First, this article presents a thorough qualitative discussion of different available technologies for push-based notifications. The aim of this discussion is to find the most suitable push-based messaging technologies for application with OGC Web Processing Services (WPS). Based on this, an event-driven architecture for asynchronous geospatial processing with the WPS is presented, building on the Web Socket Protocol as the transport protocol and the OGC Event Service as the message-oriented middleware. The proposed architecture allows pushing notifications to clients once a task has completed. This paradigm enables the efficient execution of web-based geospatial processing tasks as well as the integration of geographical analyses into event-driven real-time workflows.

1 Introduction

With location gaining increasing importance in modern web-based applications, the web is currently being transformed into a "Geospatial Web" (Scharl and Tochtermann 2007). An integral part of this development is the set of geospatial implementation and data encoding standards provided by the Open Geospatial Consortium (OGC). OGC defines a number of geospatial web standards that range from data encoding rules to data discovery services and services that enable web-based geospatial processing.

The *Web Processing Service* (*WPS*) is part of this OGC stack and offers geospatial processing capabilities within network-enabled workflows (Schut 2007) to publish and execute geo-analysis tasks (Schaeffer et al. 2012). According to the principles of service-oriented computing (cf. Erl 2008), WPS allows for reusing existing geospatial analysis algorithms instead of re-implementing them in proprietary desktop tools, which is today's common practice. As such, WPS constitutes the basis for a broad range of new application types such as, for instance, emergency support scenarios (Sagl et al. 2013) or real-time cities (Resch et al. 2012).

This article addresses two types of scenarios: analysis of large datasets and real-time workflows that process data on the fly. Typical examples for the first scenario are line

Address for correspondence: Rene Westerholt, GIScience Research Group, Institute of Geography, Heidelberg University, Berliner Straße 48, 69120 Heidelberg, Baden-Württemberg Germany. Email: rene.westerholt@geog.uni-heidelberg.de

doi: 10.1111/tgis.12104

simplification algorithms, schematization or graph-based operations like Dijkstra's shortest pathcalculation. Such operations do not allow for dividing the data into smaller chunks, because the context of the whole dataset is required for the analysis. Examples for the latter scenario type are all kinds of real-time fusion or time-critical decision support systems. In such cases of processing large datasets in complex operations, requests can potentially take hours or days to complete (Kurzbach et al. 2009). This is already challenging on desktop machines, but becomes even more problematic in networked environments that are additionally prone to communication issues (e.g. server timeouts). Therefore, WPS comes up with two different communication patterns: synchronous and asynchronous. Both are briefly explained in the following paragraphs by mentioning their particular bottlenecks for scenarios like those described above.

Like all OGC web services, WPS relies on synchronous Hyper Text Transfer Protocol (HTTP) (Min et al. 2008a). In case of the synchronous mode of WPS, communication happens in the typical request-response cycle of HTTP: (1) the client requests a resource; (2) the server processes this request; and then (3) responds with the analysis result. However, this communication paradigm is not appropriate for dynamic geospatial processing tasks involving big amounts of data and complex algorithms as described in the previous paragraph. The resulting long-lasting execution time often exceeds the maximum timeout range of HTTP servers (Resch et al. 2010), which then inevitably leads to result loss.

Consequently, the asynchronous manner of communication is better suited to the needs of complex geospatial processing tasks. It frees the client's resources during server-side processing by decoupling the request from the response. However, this mode has a critical issue when used with the OGC WPS: it relies on a polling approach (Schut 2007). Polling means constantly checking whether the results are yet available or not (Papazoglou 2008). This results in either high latency, which is inappropriate for real-time geospatial processing applications, or large amounts of overhead and network traffic (Resch et al. 2010).

To overcome these issues, we propose a push-based approach for asynchronous operation of the WPS. Instead of constantly polling the resource, we suggest pushing notifications containing the WPS operation's results to the client. This approach leads to lower overhead and network traffic (Agarwal 2012, Jiang and Duan 2012), and thus paves the way towards a more efficient kind of web-based geospatial processing. Moreover, it enables the integration of the WPS service interface with modern event-driven architectures that describe the overarching concept based on the proactive exchange of messages (Bruns and Dunkel 2010). In summary, the following research questions are addressed in this article:

1. Which technologies are suitable for enhancing WPS with pro-active notification capabilities?
2. How can these technologies and WPS be combined into one holistic approach defining a workflow and a structural sketch?
3. How can the suggested approach be integrated with the OGC WPS standard?

The remainder of this article is structured as follows: first, the state-of-the-art in the field of push-based, event-driven geospatial processing is summarized in Section 2. After this overview, some potentially appropriate technologies that could form the basis for the developed approach are qualitatively discussed in Section 3, while Section 4 contains our approach for an event-driven architecture for asynchronous geospatial processing using the OGC WPS. Section 5 deals with the integration of the approach with the current WPS specification. The results of the technology discussion as well as the approach are then critically discussed in Section 6. A proof of the concept of developed approach is provided in Section

7. Finally, a conclusion of our approach and an outlook on further research questions are given in Section 8.

2 State of the Art

The WPS issues described in Section 1 represent a long-standing problem (e.g. Diaz et al. 2010; Kurzbach et al. 2009). Resch et al. (2010) identify the usage of event-driven architectures and push-based protocols as possible solutions to overcome the problems mentioned. So far, however, no sufficient practical solution has been developed. The following subsections present a review of previous research dealing with these problems. Research from related fields like the OGC Sensor Web Enablement initiative (SWE) is analyzed (Subsection 2.1) as are some WPS-specific approaches (Subsection 2.2).

2.1 Push-Based Communication Approaches in the OGC Stack

An early push-based approach from the OGC is the Sensor Alert Service (SAS) (Simonis 2007). This service aims to realize real-time sensor data delivery and can be seen as the push-based counterpart of the Sensor Observation Service (SOS) (cf. Broering et al. 2012). It relies on the Extensible Messaging Presence Protocol (XMPP) (cf. Saint-Andre 2011) and establishes a multi-user chat (MUC) channel for every sensor. Clients that are interested in sensor data need to register to the corresponding chat room, where the sensor of interest publishes its data (Resch et al. 2010a; Foerster et al. 2009). A shortcoming of this approach is its XMPP binding and the application of a chat room, which is a roundabout way for the given purpose and adds additional complexity.

The successor of the SAS is the OGC Sensor Event Service (SES) (Echterhoff and Everding 2008). In contrast to SAS, it is not tightly coupled with a specific transport protocol. Message exchange is based on SOAP and Web Services Notification (WSN) (cf. Chappell and Liu 2006; Graham et al. 2006), which is developed by the Organization for the Advancement of Structured Information Standards (OASIS). Furthermore, SES provides more advanced filter capabilities to confine users' subscriptions (Janowicz et al. 2011; Broering et al. 2011). However, this approach comes up with a quite complex specification that goes far beyond the basic purpose of exposing notification capabilities. Therefore, it does not seem to be very suitable for the application with other OGC web services (OWS) apart from its original sensor domain.

Since the two previously mentioned approaches strongly focus on sensor data, the OGC tried to define a more general approach by developing the OGC Web Notification Service (WNS) (Simonis and Echterhoff 2007). WNS is a standalone message broker that exposes notification capabilities to other OWS. Thus, it is transport protocol-agnostic. Possible protocols can range from the Simple Mail Transfer Protocol (SMTP) to XMPP or the Short Message Service (SMS) (Jaeger et al. 2010). Although WNS is a pure notification approach, it is outdated and currently not under proactive development. A recent trend is to generalize the WNS to more general concepts. This ambition resulted in the development of the OGC Event Service (Echterhoff and Everding 2011), which is based on SOAP and WS-Notification, just like the SES. The difference to SES is the reduction to the very basics of a notification mechanism without mixing different functionalities.

The OGC recently bundled its activities on the topic of push-based data delivery into an official standard working group, called PubSub SWG (OGC 2012). This group aims to define common specifications as well as push-based enrichment of existing services.

2.2 WPS-Specific Approaches

Some event-driven approaches are specifically focussing on the WPS. Min et al. (2008) created an early approach that places a middleware in between the client and the WPS. The middleware acts like a proxy that mediates between the client and the server – no communication takes place between the client and the WPS directly. Processing tasks are propagated via subscriptions to the middleware, which then delegates the task to the WPS. The same goes with the result, which is delegated back to the client by the middleware component. After triggering process execution, the middleware continuously polls the result location and grabs the results once they are available. A problem that arises from this manner is that the polling-issue is just shifted from the client to a middleware instead of being solved.

Another approach by Jaeger et al. (2010) engages the WNS as a brokering component. This approach has been developed within an early warning scenario and aims to process sensor data. They engaged protocols that allow for a clear identification of the end user (e.g. SMS via a telephone number or email via a corresponding address). A major drawback of this approach is that it is not applicable to networked clients beyond the mentioned protocols, preventing the integration of desktop computers, networked mobile phones, etc.

An approach by Everding and Foerster (2011) uses SOAP and WS-Notification. It places an enterprise service bus between all involved components to handle all communication. Therefore, the WPS specification has been enhanced in order to implement the NotificationConsumer-interface of WS-Notification. Thus, WPS is able to receive SOAP messages directly without any message broker in between. An issue with this functional enhancement is that several different functionalities are mixed (geospatial analysis and messaging). This violates the principles of service-oriented architectures, in which every component should only cover one defined thematic scope. Furthermore, this approach makes use of several transport protocols, since it additionally uses XMPP in order to involve an OGC Sensor Alert Service. This adds additional complexity to the approach, as knowledge about different technologies is necessary to implement and use it.

The most recent approach to overcome the mentioned problems is presented by Foerster et al. (2012), which deals with processing geoinformation in real-time. Technically, their approach relies on HTTP Live Streaming (cf. Pantos and May 2013), an open-source protocol that has been established by Apple, Inc. The basic idea behind the approach is to fragment big data sets into smaller chunks and reference them on a playlist. This playlist is then continuously updated when new intermediate results become available. This approach allows providing clients with intermediate results even though the processing of the whole dataset has not yet finished. Despite the similar research direction, this approach has a different focus from the research presented in this article, since it targets the integration of continuous geo-data streams. It focuses on analysis use-cases in which small data chunks flow in gradually, such as crowdsourced geo-information from social media feeds or sensor data. In contrast, our research focuses on large data sets that must be analyzed all at once (e.g. when the datasets' topology is considered). Moreover, the approach of Foerster et al. (2012) does not focus on push-based notification, but requires the client to regularly poll the playlist. Accordingly, it provides intermediary results in real-time, but does not aim to overcome polling approaches.

3 Technology Discussion

To reach the aim of integrating WPS with event-driven architectures, some specific technologies are needed. This affects two different technological layers: on the one hand, an

underlying *push-capable transport protocol* is needed. Since an event-driven architecture is based on the proactive exchange of messages even from server to client, this task cannot be solved efficiently by using standard HTTP, as described in Section 1. Only a few protocols come into consideration for this purpose. On the other hand, a *message-oriented middleware* is needed to invoke and organize the message exchange on a logical layer. This middleware serves as a mediating component between the abstract transport layer and the applications that want to make use of it. Moreover it helps to comply with the principles of service-oriented computing by encapsulating notification capabilities into a separate component instead of being integrated into WPS directly. The following two subsections provide an evaluation of possible technologies for both areas (transport protocols in Subsection 3.1 and message-oriented middleware in Subsection 3.2).

3.1 Transport Protocols

Three different push-capable transport protocols have been evaluated qualitatively: the *Extensible Messaging Presence Protocol* (X*MPP)* (Saint-Andre 2011) and the two HTML5-based approaches *Web Socket Protocol* (Fette and Melnikov 2011) and *Server-Sent Events* (Hickson 2012). The following subsections first explain the mentioned approaches (Subsection 3.1.1), before the evaluation scheme is presented in Subsection 3.1.2, and the evaluation results are presented in Subsection 3.1.3. It should be mentioned that general performance testing is not the aim of the research described in this article. Corresponding studies on this task are referenced in the following subsections instead. The scope of our discussion is to find a technology that is most suitable for application to the needs of WPS.

3.1.1 Technology description

This subsection describes the functional details from a technological viewpoint for the three evaluated transport protocols. Those comprise XMPP, Server-Sent Events and the Web Socket Protocol. Their respective principles as well as their message formats are illustrated in Figure 1.

XMPP. XMPP is based on establishing two persistent Transport Control Protocol (TCP) connections: one for data upload from client to server and another one for data transmission towards the client. Each client is clearly referenced via a consistent ID that follows the structure of email addresses and appears in the form client@xmppserver.org. After the negotiation process is done, an XML document is set up for both connections. Every message streamed from one communication partner towards the other is appended to those documents. Consequently, those messages are XML snippets which imply a XML-based messaging protocol. When communication finishes, the XML documents are completed by adding a closing tag to each and the TCP connections are closed (Saint-Andre 2011).

Server-Sent Events. Server-Sent Events, which is part of HTML5, aims to standardize the older unstandardized COMET technologies (Zakas 2011) by introducing the so-called "EventSource" interface. This interface provides one method (*close*) and several event handlers (*onopen*, *onmessage*, *onerror*) for the exchange of data between servers and clients. Each client that wants to receive data from a server via Server-Sent Events just needs to implement the EventSource interface that is supported natively through the programming language JavaScript within most modern browser implementations (Flanagan 2012).

The principle of Server-Sent Events is the same as with the COMET technology of HTTP-streaming, which allows servers to push data to clients via the synchronous HTTP protocol. Initially, an HTTP request is sent from the client to the server, which responds with an HTTP header that designates the content with the MIME type "text/event-stream". This causes the client to keep the underlying TCP connection alive for continuous data exchange. If the connection breaks due to a HTTP server timeout, the Server-Sent Events implementation reconnects with the server-side resource automatically (Hickson 2012). The structure of the messages that are sent via Server-Sent Events consists of key-value pairs.

Web Socket Protocol. Web Socket is a technology that is also part of the HTML5 initiative. Like Server-Sent Events, Web Socket is natively supported by JavaScript within many modern browser implementations (Kyrnin et al. 2011). The protocol consists of three parts: a handshake mechanism, a message format and an Application Programming Interface (API). The handshake is needed to switch communication from HTTP to the Web Socket Protocol. This is done by sending an HTTP GET request with special headers to a Web Socket server. When this request is successful, the client and the server switch their communication from HTTP to Web Socket. The underlying TCP connection persists, instead of being cut off as at the end of a HTTP request/response cycle. After performing this handshake, messages can be exchanged in both directions over this TCP connection (Fette and Melnikov 2011).

3.1.2 Evaluation scheme

The transport protocols described in Subsection 3.1.1 have been evaluated against a defined set of criteria. On the one hand, these criteria arise from the use cases mentioned in Section 1 (long execution times, large datasets). On the other hand, the criteria have been chosen with the needs of WPS in mind. In order to differentiate these criteria according to their importance for the given purpose, they are grouped into categories to which weights are assigned. Each weight reflects the importance of the category members with respect to the given purpose:

- **Category 1:** criteria of primary importance → weight of 9
- **Category 2:** criteria of moderate importance → weight of 3
- **Category 3:** criteria of minor importance → weight of 1

Each evaluation criterion is explained separately within the following bullet points (associated categories in brackets):

- **Overhead (Category 1):** keep-alive mechanism, structural overhead, initial handshake mechanism
- **Different Clients (Category 1):** adaptability to different types of clients from a technical point of view (e.g. mobile clients or native desktop clients), including a consideration of proxy and firewall restrictions that limit accessibility
- **Diffusion and Future Prospects (Category 2):** comprehensive documentation, proactive future development, awareness amongst GI-Science community
- **Availability of Software Components (Category 2):** client-side libraries, server-side software components for realizing asynchronous communication
- **Encrypted Connections (Category 2):** available encryption mechanisms
- **Implementation Effort (Category 3):** effort that is needed for implementation

Category 1 comprises the essential requirements for asynchronous geospatial processing. The *overhead* is of critical importance for the given purpose since it is one of the main

bottlenecks of previous poll-based approaches. Moreover, the transport protocol should be applicable to a broad range of *different clients*. Otherwise, WPS would be limited in terms of its applications. The criteria in category 2 ensure *future viability* and *practicality* of the approach as well as *encryption* capabilities. Those are also important in terms of sustainability, but do not affect the aims of the approach directly, for which they are considered not to be included within category 1. Finally, category 3 subsumes a criterion of less importance (*implementation effort*) as it does not serve as an exclusion criterion with respect to the general operability of the proposed approach. However, implementation effort should also be taken into account.

The weighting of the categories is accomplished exponentially. This serves for gradating the explained differences according to the relative category importance. An exponential increase has been chosen because it reflects the outstanding importance of the higher categories better than a linear increase. The actual assessment of those categories is then performed on the basis of a maximum score of 10 points per category. The obtained number of points is multiplied by the weight of the corresponding category to form an overall assessment result per criterion.

3.1.3 Evaluation results

This subsection briefly summarizes the evaluation, lays out the evaluation rationale for each criterion, and finally presents an evaluation summary.

The *overhead* (category 1) is composed of three parts: (1) a keep-alive mechanism to prevent server-timeouts; (2) the initial handshake mechanism to negotiate the connection; and (3) the structural overhead imposed by the message design. Regarding the *keep-alive mechanism*, Web Sockets and XMPP follow similar approaches. Both send "Ping" frames (and expect corresponding "Pong" frames) to keep the persistent TCP connection alive. While Web Sockets send one Byte sized frames (Fette and Melnikov 2011), XMPP utilizes heavier XML snippets (Saint-Andre 2009). In contrast, Server-Sent Events do not provide similar behavior because they are based on HTTP with regular automatic reconnection after the server has timed out. However, these reconnections impose a significant amount of overhead since all HTTP headers need to be transferred regularly. The *handshake mechanism* is also not necessary for Server-Sent Events due to its HTTP binding. In contrast, the other two approaches switch communication to their native message formats. While Web Sockets handle this by sending one HTTP GET request (Fette and Melnikov 2011), XMPP again requires XML-based communication (Saint-Andre 2011). The *structural overhead* varies greatly among the approaches. As illustrated in Figures 1, Web Sockets are based on light-weight messages that surround the payload with just two bytes. In contrast, XMPP exchanges XML snippets. Server-Sent Events are based on a key-value-paired format that ranges inbetween (according to its amount of overhead). Gutwin et al. (2011) conducted a performance test of several real-time data transmission techniques (Comet Approaches, Java Applets and Web Sockets). They have shown that Web Sockets cause the smallest amount of overhead, the highest message send rates (nine times higher than Comet) and receive rates (14 times higher than Comet), and the lowest latency (5.8 times smaller than Comet). Similar comparisons have been undertaken by Agarwal (2012) and Jiang and Duan (2012). They found that long-polling approaches come up with a 525 times higher structural overhead than Web Sockets. In the case of small messages, the overhead of poll-based approaches can take up to five times the size of the actual message. Based on these studies, we claim Web Sockets to be the most efficient technology

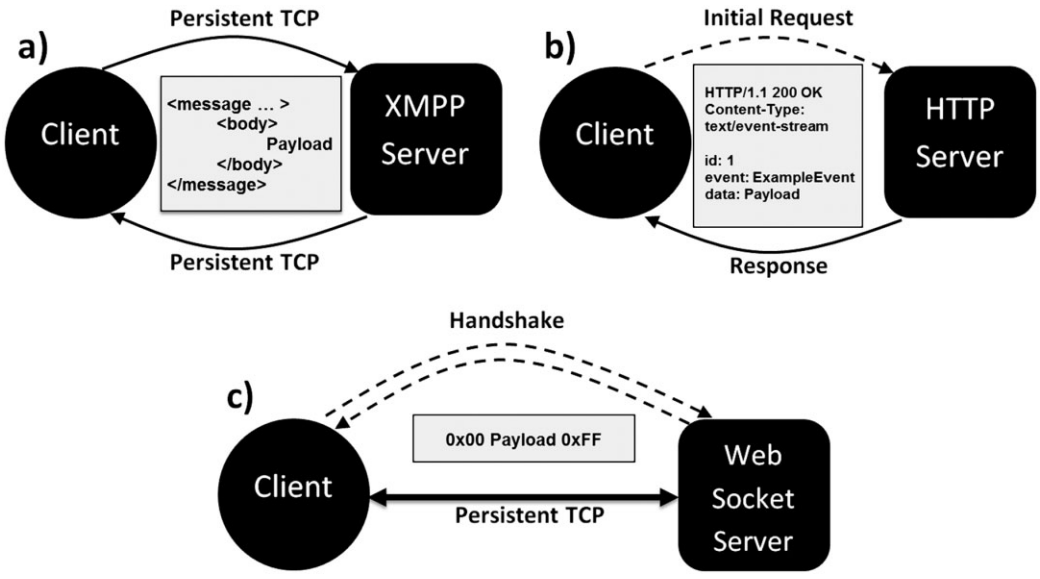


Figure 1 Principles and exemplary messages of: (a) XMPP; (b) Server Sent Events; and (c) Web Sockets

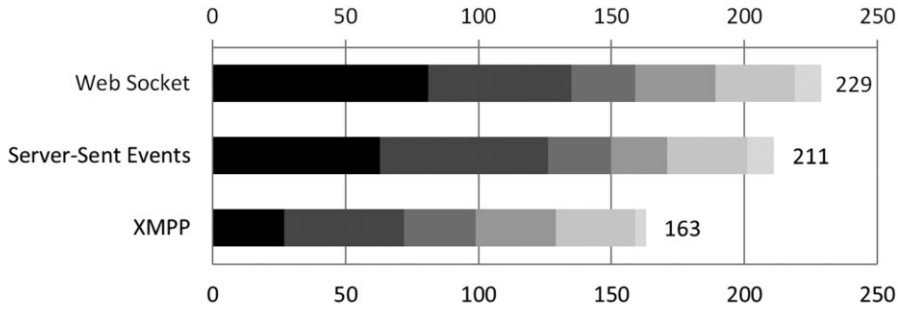
with respect to overhead. XMPP works similar, but utilizes XML-snippets. XML is heavier than the light-weight format of Web Sockets. In contrast, Server-sent Events rely on polling, which performed worst.

Another important criterion (category 1) is the possibility of the solution being integrated with a wide range of *different types of clients*. This is important because WPS is potentially usable with any kind of networked client – which should not be limited by notification-specific characteristics. The Web Socket Protocol and Server-Sent Events match this criterion best. Due to their low overhead and their underlying principles, they can serve a variety of different clients. The Web Socket Protocol and XMPP have some limitations in mobile scenarios, which originate in the usage of persistent TCP connections. Moreover, XMPP is not very suitable, since it comes up with large amounts of overhead, resulting in heavy network traffic. This limits its usability with clients that are prone to low connectivity or slow connections (e.g. mobile clients).

Other significant criteria (category 2) for the selection of a transport protocol are, e.g. the availability of software components, the diffusion and popularity (especially among the GIScience community) and the expected implementation effort.

The criterion of *availability of software components* (category 2) is best matched by the Web Socket Protocol, since many different implementations for many different purposes still exist. According to community awareness, XMPP is best known due to its application with the Sensor Alert Service (see Subsection 3.2.1). Another important aspect, particularly for safety-critical scenarios (such as military operations), is the possibility to communicate via encrypted connections (category 2). This criterion is matched well by all three protocols.

The criterion of the expected *implementation effort* (category 3) is matched sufficiently by the Web Socket Protocol and Server-Sent Events. Both provide simple APIs and are straightforward in terms of implementation.



	XMPP	Server-Sent Events	Web Socket
Overhead (90)	27	63	81
Different Clients (90)	45	63	54
Diffusion / Future Prospects (30)	27	24	24
Software Components (30)	30	21	30
Encrypted Connections (30)	30	30	30
Implementation Effort (10)	4	10	10
Total (280)	163 ≈ 58%	211 ≈ 75%	229 ≈ 82%

Figure 2 Summary of the evaluation results for transport protocols (Unit of measurement = simple score counts, maximum score in brackets, percentage of maximum noted behind the sum)

Figure 2 provides an overview of the total evaluation results for the three protocols mentioned above. It also shows that the Web Socket Protocol meets the criteria by 82% (229 out of 280 points maximum); Server-Sent Events match them by 75% (211 out of 280 points maximum) while XMPP only matches them by 58% (163 out of 280 points maximum).

3.2 Message-Oriented Middleware

Four message-oriented middleware technologies are evaluated in this subsection: three approaches by the OGC, namely the *Sensor Alert Service (SAS)* (Simonis 2007), the *Sensor Event Service (SES)* (Echterhoff and Everding 2008) and the *Web Notification Service (WNS)* (Simonis and Echterhoff 2007), and the more general *WS-Notification (WSN)* (Chappell and Liu 2006; Graham et al. 2006). Additionally, the *OGC Event Service* (Echterhoff and Everding 2011) is taken into account. However, it is evaluated together with WS-Notification, which forms its technological foundation since the OGC Event Service specification is still in the early stage of a discussion paper. The following subsections explain the mentioned technologies (Subsection 3.2.1), before the evaluation scheme is presented in Subsection 3.2.2, and the evaluation results are presented in Subsection 3.2.3.

3.2.1 Technology description

This subsection describes the functional details for the five evaluated message-oriented middleware technologies.

OGC Sensor Alert Service and OGC Sensor Event Service. The Sensor Alert Service (SAS) and Sensor Event Service (SES) both originate from the developments in the Sensor Web

Enablement Initiative (SWE). The SAS aims to notify clients about the occurrence of new event data (i.e. sensor originated data). It is built on top of a XMPP binding (see Subsection 3.1.1). The communication between clients and sensors is organized by applying multi-user chat rooms. Every transmitter that wants to publish its data forwards them to a corresponding chat room. A client registers at the SAS and subscribes for a specific multi-user chat room (Simonis 2007).

The SES can be regarded as the successor of the Sensor Alert Service. In principle, this approach is very similar to SAS in that it also acts as an information broker that forwards messages from data transmitters to clients (Echterhoff and Everding 2008). However, in many ways it enhances the concepts of the SAS by, for instance, adding advanced filter capabilities (Resch et al. 2010a). Moreover, it is decoupled from XMPP and relies on SOAP and WS-Notification instead. The actual payload needs to be defined in Observations and Measurements (O&M) format – a way to encapsulate sensor observations within the SWE initiative (Resch et al. 2010b).

Web Notification Service. Like SAS and SES, the Web Notification Service (WNS) originates from the SWE initiative. It has been introduced in addition to the SAS, but is not tightly coupled with the sensor domain. It is rather a general supplementary service that is able to augment other OGC web services with messaging capabilities. It can also be considered a protocol transducer, since it receives messages in one protocol and forwards it via another protocol (Broering et al. 2011).

WNS can act in two ways: it either performs one-way notification or it serves as a two-way communication broker. In the latter case, an asynchronous response message is expected. For both cases, WNS specifies message formats that encapsulate the original payload. Every client that wants to receive messages from a WNS needs to register first. This registration has to include an endpoint reference to which the WNS can deliver its messages. After registration, a notification transmitter is able to send messages via WNS by denoting the UserID of the recipient (Simonis and Echterhoff 2007). In many ways, WNS can be seen as an alternative draft to WS-Notification, as described in the following paragraphs. It provides similar mechanisms; but uses another terminology and is based on different protocols.

Web Services Notification and OGC Event Service. The Web Services (WS-) Notification specification from the Organization for the Advancement of Structured Information Standards (OASIS) is a very common messaging approach beyond the OGC. This approach models stateful resources in the context of HTTP-based (and thus stateless) web services. There are two different architectural patterns of how to realize WS-Notification: WS-BaseNotification and WS-Brokered-Notification (Papazoglou 2008). WS-BaseNotification (Graham and Murray 2006) provides a mechanism for peer-to-peer communication. It defines two interfaces, one to be implemented by a notification producer and the other by a notification recipient. The producer needs to handle subscriptions, while the consumer provides an operation that can be invoked by a producer when a message shall be sent to the consumer. This architectural style is particularly suitable when producers and consumers are well-known to each other and are thus tightly coupled (Papazoglou 2008). The second pattern called WS-BrokeredNotification (Chappell and Liu 2006) encapsulates the mechanisms of WS-Base-Notification into one single component.

The OGC EventService is a recent approach to generalize the OGC SAS and SES services. The thematic background of this service was the increasing demand of push-based communication in the context of OGC web services, even beyond the sensor web domain (Echterhoff

and Everding 2011). It is currently in the making and thus not yet fully defined. The EventService is based on SOAP and WS-Notification, and comprises the functionality of WS-BaseNotification within one standalone message broker component. The payload format is not restricted to any dedicated structure or type. Any textual data can be delivered (Echterhoff and Everding 2011).

3.2.2 Evaluation scheme

Analogous to the evaluation of the transport protocols (Subsection 3.1), the evaluation criteria are assigned to classes that represent their importance with respect to the given purpose and use case. The same applies to the exponential criteria weighting system. The following list provides an overview of the evaluation criteria:

- **Overhead (Category 1):** structural overhead of the messages
- **Possible Types of Notification (Category 1):** universality of the messages (no restriction to special use cases)
- **Diffusion (Category 2):** comprehensive and consistent documentation, proactive future development, awareness among GI-Science community
- **Availability of Software Components (Category 2):** availability of server-side software components
- **Implementation Effort (Category 3):** effort that is needed for implementation

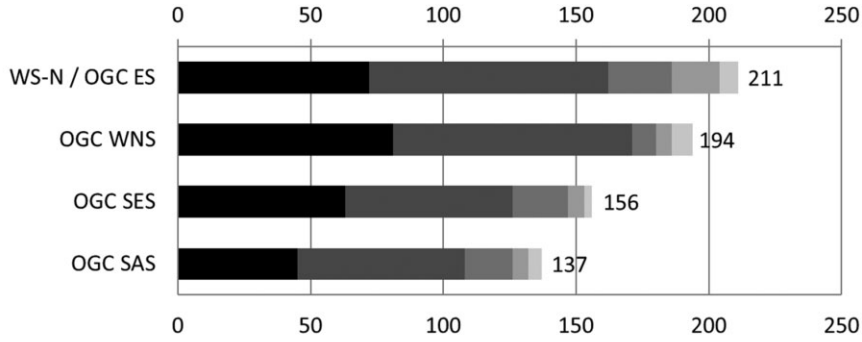
The rationale of weighting the criteria and motivating their classification follows the approach applied to the transport protocols (see Subsection 3.1.2).

3.2.3 Evaluation results

This subsection briefly summarizes the evaluation, lays out the evaluation rationale for each criterion, and finally presents an evaluation summary.

The *overhead* mainly reflects the size of the messages being exchanged. In this regard, SAS causes the largest overhead for two reasons: (1) it relies on XMPP which adds heavy XML elements; and (2) it supplies additional structural elements to describe the payload format as well as a geographic reference. The latter two points are due to the sensor-related origins of SAS, but they are unnecessary in the particular case of WPS. In contrast, SES wraps its messages in SOAP, a protocol from the web services domain. SOAP adds plenty of structural elements and vastly expands the message size. Furthermore, SES requires encoding the payload in O&M format, which might be useful for the original purpose of sensor measurements, but seems inappropriate in the given case of sending notifications. WS-Notification is also based on SOAP (since it originates from the web services domain), but it does not cause any additional overhead beyond that, as notification-related parts are completely integrated within the SOAP elements. WNS, in contrast, is not based on SOAP. It provides an independent and lightweight message format. The only additional overhead is caused by a small XML structure for describing the message transmitter.

With respect to applicability for *different types of notifications* (category 1), there are two different groups among the approaches, which are distinguished by the way they limit payload encoding. WS-Notification (and the Event Service) and WNS allow a completely independent payload definition. In contrast, SAS and SES are much more restrictive due to their original purpose. Both are highly adapted to the needs of sensor measurement provision and thus do not fit well with the aims of this article.



	OGC SAS	OGC SES	OGC WNS	WS-N / OGC ES
Overhead (90)	45	63	81	72
Different Notifications (90)	63	63	90	90
Diffusion / Future Prospects (30)	18	21	9	24
Software Components (30)	6	6	6	18
Implementation Effort (10)	5	3	8	7
Total (250)	137 ≈ 55%	156 ≈ 62%	194 ≈ 77%	211 ≈ 84%

Figure 3 Summary of the evaluation results for message-oriented middleware (Unit of measurement = simple score counts, maximum score in brackets, percentage of maximum noted behind the sum)

Diffusion among the GIScience community (category 2) depends on the origin of the approaches and the status of their proactive development. The OGC-originated approaches SAS, SES and WNS are more familiar to the GI community. However, this might be compromised by their strong and specialized focus on sensor networks. A particular disadvantage of WNS is that it is not under recent proactive development. Overall, WS-Notification is known to a much broader audience, especially beyond the GI community.

Another criterion is the *availability of software components* (category 2). While WS-Notification comes up with a broad range of implementations, the OGC approaches have only been realized in an experimental way. Since it is easier to use existing components than implementing them, WS-Notification provides the least *implementation effort* (category 3). Moreover, the other approaches (especially SAS and SES) come up with more complex specifications. The results explained above are summarized in Figure 3. It shows that WS-Notification meets the criteria by 84% (211 out of 250 points maximum); OGC WNS match them by 77% (194 out of 250 points maximum); OGC SES match them by 62% (156 out of 250 points maximum) while OGC SAS only matches them by 55% (137 out of 250 points maximum).

3.3 Evaluation Summary and Design Decisions

Based on the results of the transport protocol evaluation (Subsection 3.1.3), the **Web Socket Protocol** has been determined to be the best choice. This is due to the lowest amount of overhead, a big

number of available software components, and a quite low effort for implementation. In contrast, XMPP by far causes the largest overhead, which is due to its XML binding. As a result from this, it is not well-suited for many clients like mobile ones. Server-Sent Events reside between the two other protocols.

Following the message-oriented middleware evaluation (Subsection 3.2), **WS-Notification (WSN)** is the best-suited approach for the given purpose. Since it is completely content-agnostic, it can serve most types of notifications and is very popular among many different domains. The only issue with WSN is its mandatory SOAP-binding, which adds some additional overhead. The OGC WNS shows acceptable evaluation results as well, but is outdated and not being further developed. SES and SAS are tightly coupled with the sensor domain and are characterized by a large implementation effort (due to their complex specifications). Therefore these two approaches are disregarded for the sake of this article.

To conclude, the technology stack is made up of the *Web Socket Protocol* as the underlying technology and *WS-Notification* as the message-oriented middleware component. In order to incorporate the most recent OGC approach for this purpose, the WSN-based OGC *Event Service* is used in the architecture proposed in Section 4.

4 Architecture

In order to combine the investigated technology stack (see Section 3) with the WPS specification, these elements are integrated into a common architecture. Although the architecture is built on top of the technology stack from Section 3, it is not tightly coupled with it, i.e. the architecture is technology-agnostic. This enables the simple substitution of single components, which is important to comply with potential future developments. Subsection 4.1 describes the architectural structure of the proposed solution. Afterwards, Subsection 4.2 presents a workflow for asynchronous WPS applications that represents the logical sequence of actions.

4.1 Structure

The architecture of the presented solution is made up of several distinct components. The ones that are essential for the actual approach are the *Event Service* and the *Web Socket server*. Both form the core part of the notification mechanism and map the technology stack from Section 3 onto the approach suggested within this article.

The *Event Service* acts as a message broker between the WPS and underlying communication technology. It forwards messages from the WPS to the correct endpoint. The protocols used for this purpose are WS-Notification and SOAP (see Subsection 3.2). Furthermore, the *Event Service* manages clients' subscriptions, which is done via an additional service instance that is called *SubscriptionManager*. This part is instantiated when a client subscribes and additionally serves for unique client identification. The reason for using a separate message broker instead of integrating its functionality directly into the WPS is to keep both functionalities separated. Mixing geospatial processing capabilities and messagingfunctionality within one component would disrupt the requirements of service-oriented computing. In contrast, the chosen design allows reusing the Event Service's functionality even beyond WPS. Any other service being deployed on the same server could also make use of it.

In order to push messages to the client, a suitable endpoint is needed. This is accomplished by a *Web Socket server*, which implements the Web Socket Protocol explained in Subsection

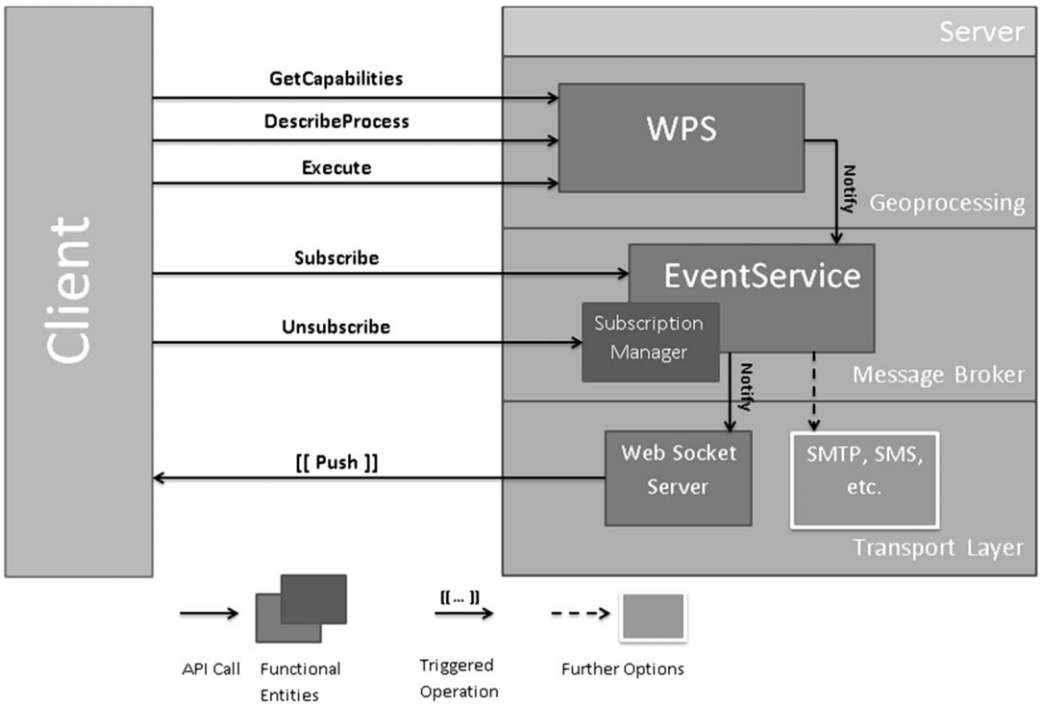


Figure 4 Architectural sketch of the proposed approach

3.1. The *Web Socket server* receives notifications from the *Event Service* and pushes them to the correct recipient. In order to be applicable to a wide range of application scenarios, the architecture concludes further potential endpoints that can serve other protocols as well. The architecture described above is visualized in Figure 4.

The architectural structure of the approach (see Figure 4) comprises several different design decision details. One example is the way the clients' subscription is performed. This could be done in two different ways: either the WPS subscribes the client at the *Event Service* or the client subscribes on its own. The second option has been chosen for the approach presented in this article. This has two advantages: There is less complexity on the server side, which enhances integrability of the approach into existing WPS implementations. Furthermore, this solution enables easier integration with other OGC web services. The latter aspect should be kept in mind to avoid monolithic development.

Furthermore, it should be noted that the client does not subscribe for a specific topic (i.e. a specific geo-computation result in this case). The reason for this is that the subscription takes place before process execution. The major advantage that arises from this behavior is that one subscription can be reused for several distinct processing tasks. This results in a reduction of consumed network and server resources, since fewer subscriptions and persistent connections must be managed.

Another important design decision concerns the way of coupling the WPS with the *Event Service*. Both could have been combined into one component in order to form an "enhanced WPS". This would have saved one additional network call (notify request from WPS to Event Service). However, tightly coupling those two components (i.e. defining a hard-wired workflow) would make a substitution of one of them later on very difficult. This point gains even more impor-

tance when considering the immature state of the Event Service. Moreover, combining two functionally distinct components into one service does not comply with the rationale of service-oriented architectures (SOA).

From a technical viewpoint, the communication endpoints could also have been integrated into the Event Service. Then they would have been fixed parts of the message broker. However, in order to keep the architecture extensible, they are designed as fully-fledged standalone web services. This enables them to be used by other software components as well as the WPS. Furthermore this design enables an easy addition of additional endpoints like SMTP or SMS servers.

4.2 Workflow

The architecture described in Subsection 4.1 translates into a sequential workflow. The workflow basically extends the standard sequence as defined in the WPS specification.

In a first step, the client requests the service's metadata from the WPS. These metadata contain the URL of the Event Service, a list of all provided communication endpoints (i.e. supported protocols), information about whether a process supports notification or not, etc. Based on this information, the client is able to subscribe at the Event Service. Therefore, the Event Service creates a so-called *SubscriptionManager* object. This component is required by the WS-Notification specification and acts like a standalone service, which is responsible for the administration of the clients' subscriptions. However, it is not a standalone service from a physical point of view: the separation only happens on a logical layer (see Graham et al. 2006 for further information).

After performing these preparatory steps, the actual asynchronous process execution can happen. Here, the client needs to pass some of the metadata it received before to the *Execute* request. Furthermore, the client needs to add the desired protocol and the message broker URL. Additionally, the URL of the SubscriptionManager needs to be sent, since it serves for a clear allocation of the client to the Event Service. The response that arrives from the WPS then contains a URL of a Web Socket resource.

Thereafter, the client has to establish a connection to this Web Socket resource. Then, it can await the notification by the server that signals the process completion. When the client has received this notification, it can disconnect from the Web Socket resource, unsubscribe at the SubscriptionManager (i.e. at the Event Service), and retrieve the processing result from the URL contained in the earlier *Execute* response.

Figure 5 visualizes the depicted workflow. It contains the standard sequence as defined by the WPS specification as well as the additional parts defined within this approach. The latter parts are shaded in grey.

5 Integration with the WPS Standard

The architecture described in Section 4 defines the way, in which the components need to be arranged to form an event-driven architecture supporting asynchronous geospatial processing. For broad applicability, the architectural components (and the event-driven behavior) need to be integrated with the existing WPS standard. This section presents an approach to integration, which aims at simple integratability without having to modify the standard's core, only extending it by a few elements. The following three subsections (5.1–5.3) discuss the three

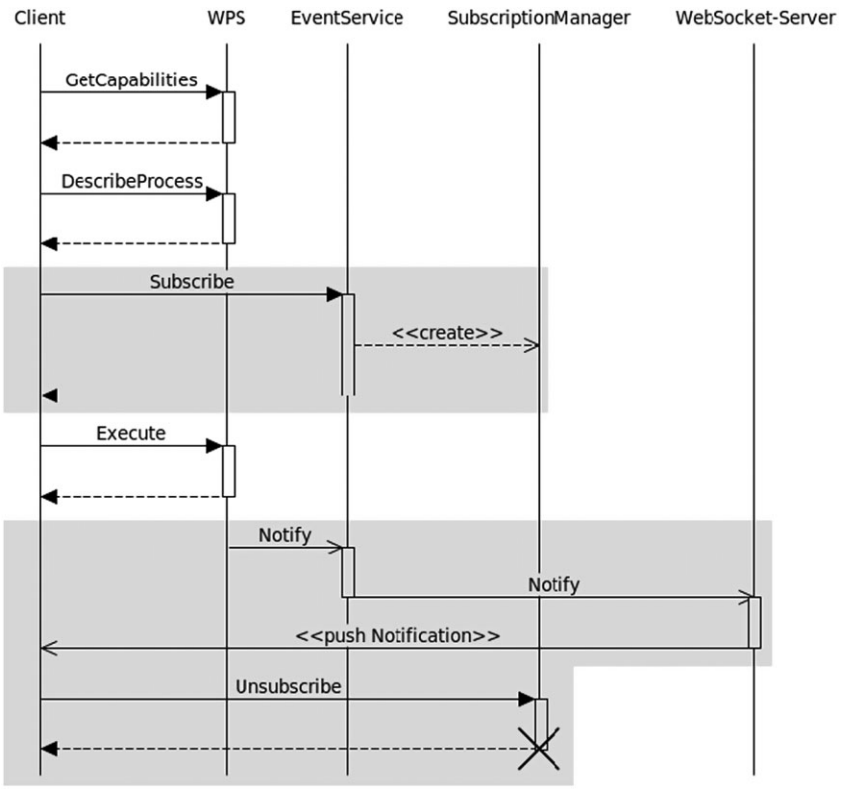


Figure 5 Sequence diagram that represents the proposed workflow (additional parts beyond standard WPS shaded in grey)

WPS operations that need to be augmented – *GetCapabilities*, *DescribeProcess* and *Execute*. Finally, Subsection 5.4 proposes a suitable notification message format.

5.1 GetCapabilities Operation

According to the workflow from Subsection 4.2, a client first needs to request the WPS's capabilities. Since this operation is the first contact point of the client, it needs to provide basic information on the procedure of push-based processing. This comprises at least a representation of the message broker (i.e. the Event Service) and a list of provided notification protocols. These two aspects are essential to start the notification process. In order to integrate this information into the WPS's capabilities response, they are encapsulated within a separate XML structure that represents the message broker. For this purpose we suggest introducing respective XML tags (*MessageBroker*, *CommunicationProtocols* and *CommunicationProtocol*) to the OWS namespace. Such a structure could provide the URL for subscription and the mentioned list of offered protocols. The following code-box gives an example for the proposed structure (proposed elements are highlighted by bold font):


```
<wps:Capabilities>
      . . .
      <ows:MessageBroker>
            <ows:Operation name= Subscribe >
                  <ows:DCP>
                        <ows:HTTP>
                              <ows:Post xlink:href= http://path-to-EventService />
                        </ows:HTTP>
                  </ows:DCP>
            </ows:Operation>
            <ows:CommunicationProtocols>
                  <ows:CommunicationProtocol type= WebSocket />
                  [[additional protocols ]]
            </ows: CommunicationProtocols >
      </ows:MessageBroker>
</wps:Capabilities>
```

5.2 DescribeProcess Operation

In contrast to the general information given by the *GetCapabilities* operation, the *DescribeProcess* operation provides process-specific information. For the purpose of asynchronous geospatial processing, this effectively means an attribute indicating the support status of notifications for a specific process. Therefore, an additional parameter "*notificationSupported*" is integrated into the root element of the process description. This is in alignment with the already applied concepts for status reporting ("*statusSupported*") and result storage ("*storeSupported*"). The additional parameter is necessary in order to not avoid the existing mechanism without proactive notification. For some types of processing, e.g. tasks that take only small amounts of time, this could be the more suitable option. The following code-box provides an example for a root element augmented by the proposed parameter:

```
<ProcessDescription
      wps:processVersion="1.0.0"
      statusSupported="true"
      storeSupported="true"
      notificationSupported= true
>
```

5.3 Execute Operation

For operating WPS in asynchronous mode using notifications, a "trigger" needs to be defined. This has to be sent to the client when a process has finished. Therefore, the *Execute* operation needs to be augmented in two different ways: (1) the *Execute* request needs to be enhanced with all necessary information to guarantee correct notification delivery; and (2) additionally, the response of this request also needs to be extended in order to allow the client to connect to the Web Socket resource. The latter aspect is crucial for successfully receiving the notification.

The proposed enhancement of the request contains the following additional parameters: "*processFinishedNotification=TRUE*" (indicator whether to use notification or not),

"broker=http://. . ." (the URL of the message broker), "subscriptionManager=http://. . ." (the ID of the client at the *Event Service*), and "*notificationProtocol=WebSocket*" (the protocol to be used for notification delivery). The following box shows an example that is integrated into a HTTP POST based *Execute* request. The corresponding HTTP GET request is similar to this example and contains the same parameters as key-value pairs.

```
<wps:ProcessFinishedNotification
        broker= http://path-to-Broker
        subscriptionManager= http://path-to-subscriptionManager
        notificationProtocol= WebSocket >
              TRUE
</wps:ProcessFinishNotification>
```

In addition, the *Execute* response is augmented by an XML structure representing the notification resource (here: a Web Socket server) to connect to. In order to provide the client with all necessary information, this structure contains the type of protocol, the URL of the corresponding resource and a unique result ID. The latter is essential because a client could perform multiple analysis tasks at the same time. In order to enable the client to distinguish between the different awaited results, this result ID is necessary. The subsequent box gives an example of the proposed structure:

```
<wps:NotificationResource
        protocol= WebSocket
        uri= ws://resource
        resultID= 1
/>
```

5.4 Notification Message Format

Since the exchange of messages is essential to event-driven architectures, the design of the message format is crucially important within this approach. The proposed format consists of two different parts: a content-neutral framework and a use case specific payload. Since it does not make sense to define a message format exclusively for usage with the WPS, the goal is to create one common message format that can potentially be used across the whole OGC stack. Therefore, the message frame has to meet the following two criteria:

- It needs to be able to carry any textual information.
- It needs to be self-describing.

These two criteria guarantee the possibility of using the message format across different application scenarios. In contrast, the payload is usually use case specific. In the given case, this is the notification about process completion. Thus, the appropriate payload needs to contain all the information necessary for that specific use case. The payload defined in this approach is created against this background. Therefore, it models the specific use-case of process completion.

The proposed message format is composed of two already existing structures. The frame is derived from the message format that has been proposed within the OGC Web Notification Service best practice paper (cf. Simonis and Echterhoff 2007). The payload is derived from a structure provided by the WPS specification itself. The *<wps:Status>* structure

(cf. Schut 2007, Table 55), which is used for the "statusSupported" option in WPS. This also fits the given use case since it provides a tag that represents successful process completion.

These two components are able to meet the criteria given above. The only parameter that has been added to the payload is the *resultID*. This information is important because a client could potentially trigger different geospatial processes at the same time as explained in the previous subsection. The following box visualizes an example of a complete notification message:

```
<ows:NotificationMessage xmlns:ows= http://. . . >
    <ServiceDescription>
        <ServiceType>WPS</ServiceType>
        <ServiceTypeVersion>1.0.0</ServiceTypeVersion>
        <ServiceURL>http://path-to-WPS</ServiceURL>
    </ServiceDescription>
    <Payload xmlns:wps= http://www.opengeospatial.net/wps >
        <wps:Status creationTime= 2007-04-18T. . .  resultID= 1 >
            <wps:ProcessSucceeded/>
        </wps:Status>
    </Payload>
</ows:NotificationMessage>
```

6 Discussion

The following two subsections provide a critical discussion of the results that have been presented in terms of the proposed architecture, the workflow and the extension of the current WPS specification. This discussion contains advantages and disadvantages concerning the incorporated technologies as well as the investigated holistic approach.

6.1 Incorporated Technologies

After thorough evaluation of available transport protocols, the *Web Socket Protocol* has been chosen to serve as the most suitable push-capable protocol. Web Socket is characterized by a low amount of overhead, which is due to the light-weight design of the exchanged data frames and to its operation beyond HTTP polling (thus minimizing the number of HTTP connection establishments and thus exchanged headers). Furthermore, it is quite simple to use and implement. Another advantage is its native integration into JavaScript within most modern browser implementations. Even beyond the browser clients, there are many different client libraries and server products available.

However, Web Socket has a few disadvantages, too. First, this technology is not well suited to mobile clients who are potentially exposed to unstable connectivity as Web Sockets rely on persistent connections. When connections break too frequently, the overhead for reconnecting may grow quite large. Another problem occurs when a client explicitly interrupts the connection (e.g. when the machine is shut down). Then, the notification is lost and the client falls back to regularly polling the result location. A possible way to overcome this problem is the application of the Web Storage technology (cf. Hickson 2011) from the

HTML5 initiative. With this technology, clients are able to store all relevant information about the Web Socket connection persistently, even when the machine is shut down.

Another issue with Web Sockets occurs when intermediaries like proxy servers are situated between the client and WPS. The handshake mechanism contains several headers that are meant to be hop-by-hop headers within the HTTP/1.1 specification (cf. Fielding et al. 1999, section 13.5.1). This means that those headers are not forwarded through the network topology, but cut off at the first intermediary instead. This would prevent the connection from being established. Last but not least, the use of persistent connections causes some administrative overhead at the server, e.g. for keeping the connections alive. Furthermore, there are often restrictions on the maximum number of allowed simultaneous connections.

Most of the mentioned limitations are inevitable when dealing with the common established Internet infrastructures. Since clients are not directly addressable by default, push-based communication can only be achieved either by engaging frequent polling or by maintaining persistent connections. However, the presented solution seems to be significantly better (in terms of overhead, network consumption, etc.) than comparable poll-based approaches.

Apart from the transport protocol, the OGC *Event Service* forms the other component of the proposed technology stack. The *Event Service* serves as a message-oriented middleware. It is based on SOAP and therefore comes up with some additional structural overhead. That leads to higher consumption of network resources and to higher parsing effort. The usage of SOAP should be reconsidered in terms of its suitability and necessity for the OGC stack in general. This is particularly important since RESTful (Representational State Transfer) services are rapidly gaining importance. Such approaches come up with low implementation effort and small communication overhead (Yazar and Dunkels 2009). However, for now XML-based request/response patterns are state-of-the-art for OGC Web Services.

6.2 Architecture and Integration

The investigated asynchronous geospatial processing architecture has been designed with respect to loose coupling of the components. This enables simple substitution of single components and makes the architecture flexible for the integration of future developments. Furthermore, the architecture is extensible in terms of adding additional communication endpoints, serving different communication protocols. Thus, the architecture is not tightly coupled with the technology stack described in Section 3. Another particularity in the architectures' design is the possibility of applying it to other OGC web services beyond WPS. Other services such as WFS, WMS and several sensor-related approaches could be used instead of WPS at low effort.

A disadvantage of the current design is a lack of error-handling capabilities. When the WPS encounters an error, it writes a corresponding message to the result location. If no notification is sent due to the error, the client keeps listening without noticing that processing has failed. The reason is that the client does not constantly check the result location and is therefore not aware of the error that occurred. One possible solution to this problem is to require a notification with status "*wps:ProcessFailed*" when an error occurs in the processing operation.

7 Proof of Concept: Prototypical Implementation

In order to prove the applicability of our proposed approach, we developed a prototypical implementation. With this implementation we intend to show that our approach works in

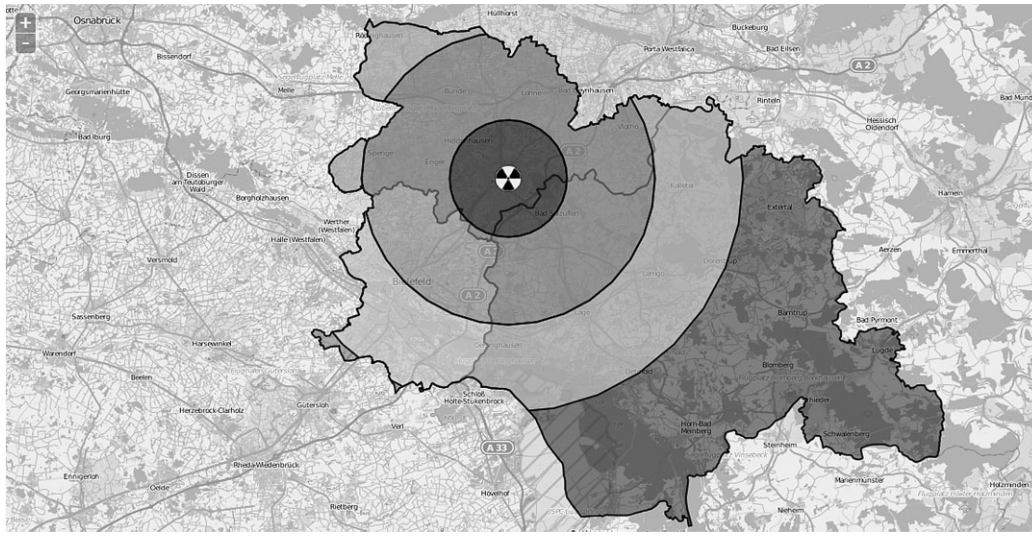


Figure 6 Illustration of our test scenario with severity codes shaded in different colours

terms of technology integration, the combination of WPS with push-based event services and the integration of the message broker with the transport layer protocol. Furthermore, our basic implementation shows that implementing the proposed architecture is possible with reasonable effort (it took us approximately eight hours of programming and infrastructure setup). This implementation does *not* aim:

1. To measure network throughput (this has been done by several other studies; see Subsection 3.1.1);
2. To implement a sophisticated analysis task for assessing performance measures; or
3. To provide a fully-fledged software solution.

The latter means that we do not enrich the capabilities of the WPS by our proposed additions. This would be little effort from an engineering viewpoint, but would involve changing an existing WPS implementation (GeoServer). This cannot be done with acceptable effort, given the low relevance of this aspect for proving the overall concept.

The hypothetical situation we have chosen for our implementation is as follows: stakeholders from several administrative districts are interested in the calculation of different evacuation zones around a nuclear power plant in case of a nuclear accident. These evacuation zones are discriminated into three different severity codes (see Figure 6), depending on the distance to the disaster site. According to the severity codes, two consecutive geospatial processing steps have to be performed:

1. Three subsequent buffering operations using different radii have to be performed on the location of the power plant; and
2. The results from the buffering process need to be intersected with the respective administrative district borders.

Even though the scenario is not a usual use case for long-running processes (cf. Section 1), it still proves the feasibility and the usefulness of the approach presented in this article as it

demonstrates the concrete realization of an event service and a Web Socket server, the implementation of two test processes, and the integration of the results in a browser-based map client.

For the concrete implementation of the components shown in Figure 4, two existing technologies have been used to establish the basic service-oriented architecture that leverages the request/response model: Apache Tomcat (Apache 2013) (the server component and the Java runtime environment) and GeoServer (OpenGeo 2013) (the map server and the OGC client). A major advantage of using Apache Tomcat is that it includes a ready-to-use Web Socket library. This can be leveraged for implementing a Web Socket server.

For the WPS component, the GeoServer WPS extension has been used in the demonstration. The two-step algorithm explained above has been implemented in Java. In case of successful calculation of the results, they are stored on the server while a notification is sent to the Event Service.

The Event Service, which is necessary to register the client and to notify the client via the Web Socket Server (see Figure 4), has been implemented as a Java Servlet (see Hunter and Crawford 2010). It can handle client registration, message forwarding and deregistration. The administration of client subscriptions is handled by incorporating SQLite (SQLite Consortium 2013), a file-based small-footprint database system. In the notification workflow, the Event Service serves as a message forwarding component that receives notifications from the WPS and forwards them to the Web Socket server by adding metadata about the correct recipient (i.e. client).

Finally, the Web Socket server, which pushes notifications to the client, has been realized as a Java Servlet using Tomcat's Web Socket library. The server is able to recognize the incoming requests for performing the Web Socket handshake and returns a representation of the connection to the client. When a notification is forwarded to the Web Socket server, it is queued into a client-related message queue. This queue then is processed by sequentially dequeuing its elements following a "First in first out" (FIFO) strategy. This enables to process a high amount of notifications in short time intervals without losing any of them.

On the client side we implemented a simple JavaScript client that is able to send requests to the Event Service and the WPS. First, the client registers with the Event Service using an Asynchronous JavaScript and XML (AJAX) request and then sends an "*execute*" request to the WPS to call the operations as described above (buffering and intersecting). After the response to this request has been received, the Web Socket connection needs to be established since the client also needs to be aware of incoming messages. This connection remains idle until the awaited notification occurs.

In the current implementation, the Web Socket server is able to receive incoming messages and push them to the listening clients. That server was implemented by inheriting from a superclass called "*WebSocketServlet*". This class is provided by Tomcat's Web Socket library and allows the overriding of some methods for broadcasting information to clients as well as reacting to incoming messages. In this way, the Web Socket server can be assigned use-case specific behavior. The extension of the WPS operation metadata as described in Section 5 has not been implemented, as this would require too much effort for a demonstration. However, it is not essential for the proof of our concept.

Our demonstration shows that the approach presented in this article works well using existing technologies and the architecture can be implemented with reasonable effort. The proposed architecture allows for pushing notifications to clients when a task has completed. Furthermore, it enables the efficient execution of web-based geospatial processing tasks as well as the integration of complex geographical analyses into event-driven, real-time workflows.

This integrability is further increased by the use of HTML5 standard technologies. The test application demonstrates this by using standard browsers (i.e. Firefox, Internet Explorer (version 10 or higher)) for testing.

8 Conclusions

Geospatial processing tasks like solar potential analyses or floodplain investigations within flood scenarios are often complex and deal with large amounts of data. If such analyses are performed in distributed web-based systems, technical capabilities are mostly not sufficient. Major problems are the potentially long execution times and the vast amount of messaging overhead that arise from common poll-based approaches. To overcome these issues, an approach for an event-driven architecture for web-based geospatial processing has been proposed within this article.

The proposed architecture consists of two parts: the architectural structure and a corresponding workflow. The parts that are comprised by the architecture include the client, the WPS, a message broker and some communication endpoints.

The approach builds on a thorough discussion of different available technologies for push-based notifications. Based on this discussion, an event-driven architecture for asynchronous geospatial processing with the OGC Web Processing Service has been developed building on the *Web Socket Protocol* as the transport protocol and the *OGC Event Service* as the message-oriented middleware. All involved parts of the structure are designed to be fully-fledged web services and can easily be substituted, i.e. they are loosely coupled. Furthermore, the whole architecture is designed to be technology-agnostic. This means that neither the suggested transport protocol (*Web Socket*) nor the proposed message broker (*OGC Event Service*) are mandatory by default.

The integration of the architecture with the WPS provides metadata enhancement and an appropriate message format. The general WPS capabilities are enriched with information about the message broker and available communication endpoints, the specific process descriptions are extended by a parameter that specifies the processes notification support, and the execution operation is enriched by a trigger mechanism to invoke notification. The message format consists of a content-neutral framework and a payload that represents a status report about process completion. Like the architecture, these parts are also designed to be content-agnostic.

Our demonstration shows that the approach presented in this article works well using existing technologies and the architecture can be implemented with reasonable effort. The concrete implementation comprises a WPS (the existing GeoServer WPS has been used), an Event Service, a Web Socket server, and a JavaScript-based client application that sends requests to the WPS, communicates with the Event Service, and displays the results on a simple map. It has been shown that the developed architecture allows for pushing notifications to clients when a task has completed. Furthermore, it enables the efficient execution of web-based geospatial processing tasks as well as the integration of complex geographical analyses into event-driven, real-time workflows.

In conclusion, the approach presented by this article raises some further research questions:

- How can other types of OWS be integrated with the suggested architecture?
- What could a common message format for the whole range of OGC web services look like?

- How are RESTful services more appropriate to use cases involving push-based communication than approaches using SOAP?
- How can the approach be applied to more use case scenarios (like mobile ones)?

References

Agarwal S 2012 Real-time web application roadblock: Performance penalty of HTML sockets. In *Proceedings of the IEEE Communication QoS, Reliability and Modelling Symposium*, Ottawa, Ontario

Apache 2013 Apache Tomcat Project. WWW document, http://tomcat.apache.org/

Broering A, Echterhoff J, Jirka S, Simonis I, Everding T, Stasch C, Liang S, and Lemmens R 2011 New generation sensor web enablement. *Sensors* 11: 2652–99

Broering A, Stasch C, and Echterhoff J 2012 *OGC Sensor Observation Service Interface Standard*. Wayland, MA, Open Geospatial Consortium Implementation Standard (available at https://portal.opengeospatial.org/files/?artifact_id=47599)

Bruns R and Dunkel J 2010 *Event-Driven Architecture*. Berlin, Springer

Chappell D and Liu L 2006 Web Services Brokered Notification 1.3: Implementation Standard. WWW document, http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.htm

Diaz L, Pepe M, Granell C, Carrara P, and Rampini A 2010 Developing and chaining web processing services for hydrological models. In *Proceedings of the First ISPRS International Workshop on Pervasive Web Mapping, Geoprocessing and Services (WebMGS 2010)*, Como, Italy

Echterhoff J and Everding T 2008 *OpenGIS Sensor Event Service Interface Specification*. Wayland, MA, Open Geospatial Consortium Discussion Paper (available at http://portal.opengeospatial.org/files/?artifact_id=29576)

Echterhoff J and Everding T 2011 *OGC Event Service: Review and Current State*. Wayland, MA, Open Geospatial Consortium Discussion Paper (available at https://portal.opengeospatial.org/files/?artifact_id=45850)

Erl T 2008 *SOA: Principles of Service Design*. Upper Saddle River, NJ, Prentice Hall

Everding T and Foerster T 2011 An event-driven architecture for decision support. In *Proceedings of Geoinformatik 2011: Geochange*, Muenster, Germany

Fette I and Melnikov A 2011 RFC 6455: The Web Socket Protocol (Proposed Standard). WWW document, http://tools.ietf.org/html/rfc6455

Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P, and Berners-Lee T 1999 Hypertext Transfer Protocol: HTTP/1.1 (Implementation Standard). WWW document, http://www.w3.org/Protocols/rfc2616/rfc2616.html

Flanagan D 2012 *JavaScript Pocket Reference*. Sebastopol, CA, O'Reilly

Foerster T, Schaeffer B, Jirka S, and Brauner J 2009 Integrating web-based sensor information into geospatial mass-market Applications through OGC web processing service. *International Journal on Advances in Intelligent Systems* 2: 278–87

Foerster T, Baranski B, and Borsutzky H 2012 Live geoinformation with standardized geoprocessing service. In Gensel J, Josselin D, and Vandenbroucke D (eds) *Bridging the Geographic Information Sciences*. Berlin, Springer Lecture Notes in Geoinformation and Cartography: 99–118

Graham S, Hull D, and Murray B 2006 Web Services Base Notification 1.3: Implementation Standard. WWW document, http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.htm

Gutwin C A, Lippold M, and Graham T C N 2011 Real-time groupware in the browser: Testing the performance of web-based networking. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, Hangzhou, China

Hickson I 2011 Web Storage: Proposed Standard. WWW document, http://www.w3.org/TR/webstorage/

Hickson I 2012 Server-Sent Events: Proposed Standard. WWW document, http://www.w3.org/TR/2012/WD-eventsource-20120426/

Hunter J and Crawford W 2010 *Java Servlet Programming*. Sebastopol, CA, O'Reilly

Jaeger S, Pauls H, Mayer C, Huber B, Dietz R, Greve K, and Camek T 2010 Informationstechnik in der Frühwarnmodellierung. In Bell R, Mayer J, Pohl J, Greiving S, and Glade T (eds) *Integrative Frühwarnsysteme für gravitative Massenbewegungen: Monitoring, Modellierung, Implementierung*, Essen, Germany, Klartext Verlag: 155–179

Janowicz K, Broering A, Stasch C, Schade S, Everding T, and Llaves A 2011 A RESTful proxy and data model for linked sensor data. *International Journal of Digital Earth* 4: 1–22

Jiang F and Duan H 2012 Application research on web socket technology on web tree component. In *Proceedings of the International Symposium on Information Technology in Medicine and Education*, Hokodate, Japan

Kurzbach S, Pasche E, Lanig S, and Zipf A 2009 Benefits of grid computing for flood modeling in service-oriented spatial data infrastructures. *GIS.Science: Die Zeitschrift für Geoinformatik* 3: 89–97

Kyrnin J, Hudson C, and Leadbetter T 2011 *The HTML5 Developer's Collection*. Upper Saddle Creek, NJ, Pearson

Min M, Chen N, Di L, Yu G, and Gong J 2008a Augmenting the OGC web processing service with message-based asynchronous notification. In *Proceedings of the IEEE Geoscience and Remote Sensing Symposium (IGARSS 2008)*, Boston, Massachusetts: 1337–40

Min M, Di L, Yu G, Chen N and Gong J 2008b Asynchronous OGC web services mechanisms. In *Proceedings of the International Conference on Earth Observation Data Processing and Analysis*, Wuhan, China

OGC 2012 PubSub SWG. WWW document, http://www.opengeospatial.org/projects/groups/pubsubswg

OpenGeo 2013 GeoServer Project. WWW document, http://www.geoserver.org/

Pantos R and May W 2013 HTTP Live Streaming: Informational Internet-Draft. WWW document, http://tools.ietf.org/html/draft-pantos-http-live-streaming-11

Papazoglou M P 2008 *Web Services: Principles and Technology*. Harlow, UK, Pearson Education

Resch B, Blaschke T, and Mittlboeck M 2010a Live geography: Interoperable geo-sensor webs facilitating the vision of Digital Earth. *International Journal on Advances in Networks and Services* 3: 323–32

Resch B, Mittlboeck M, and Lippautz M 2010b Pervasive monitoring: An intelligent sensor pod approach for standardised measurement infrastructures. *Sensors* 10: 11440–67

Resch B, Sagl G, Blaschke T and Mittlboeck M 2010c Distributed web-processing for ubiquitous information services: OGC WPS critically revisited. In *Proceedings of the Sixth International Conference on Geographic Information Science (GIScience 2010)*, Zurich, Switzerland

Resch B, Zipf A, Breuss-Schneeweis P, Beinat E, and Boher M 2012 Towards the live city: Paving the way to real-time urbanism. *International Journal on Advances in Intelligent Systems* 5: 470–82

Sagl G, Resch B, Mittlboeck M, Hochwimmer B, Lippautz M, and Roth C 2013 Standardised geo-sensor webs and web-based geo-processing for near real-time situational awareness in emergency management. *International Journal of Business Continuity and Risk Management* 3: 339–58

Saint-Andre P 2011 RFC 6120: Extensible Messaging and Presence Protocol (XMPP) (Core Proposed Standard). WWW document, http://tools.ietf.org/html/rfc6120

Saint-Andre P 2009 XEP-0199: XMPP Ping. WWW document, http://xmpp.org/extensions/xep-0199.html

Schaeffer B, Baranski B, Foerster T, and Brauner J 2012 A service-oriented framework for real-time and distributed geoprocessing. In Bocher E and Neteler M (eds) *Geospatial Free and Open Source Software in the 21st Century*. Nantes, France, Springer Lecture Notes in Geoinformation and Cartography: 3–20

Scharl A and Tochtermann K 2007 *The Geospatial Web: How Geo-Browsers, Social Software and the Web 2.0 are Shaping the Network Society*. Berlin, Springer

Schut P 2007 *OpenGIS Web Processing Service*. Wayland, Open Geospatial Consortium Implementation Standard (available at http://portal.opengeospatial.org/files/?artifact_id=24151)

Simonis I 2007 *OGC Sensor Alert Service*. Wayland, MA, Open Geospatial Consortium Best Practice Paper (available at http://portal.opengeospatial.org/files/?artifact_id=15588)

Simonis I and Echterhoff J 2007 *Draft OpenGIS Web Notification Service Implementation Specification*. Wayland, MA, Open Geospatial Consortium Best Practice Paper (available at http://portal.opengeospatial.org/files/?artifact_id=18776)

SQLite Consortium 2013 SQLite Project. WWW document, http://www.sqlite.org/

Yazar D and Dunkels A 2009 Efficient application integration in IP-based sensor networks. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, Berkeley, California

Zakas N C 2011 *Professional JavaScript for Web Developers*. Indianapolis, IN, John Wiley and Sons

