Towards a Solution for the Public Web-Based GIS Monitoring and Alerting System

Jitka Hübnerová

Abstract This paper deals with the issue of interoperability of heterogeneous sensor systems and the availability of their data from a global perspective. We show the application of previously developed WEDA architecture style into a GIS based experimental system and we present the performance analysis results of the system. The paper also presents its strengths as being a firewall-friendly, web-standards based solution that can be plugged into existing applications without needing to completely rewrite them (which is good when using OGC Sensor Web Enablement services). The paper compares the new style with styles which are used today in OGC Webservices. We then present an alpha version of the experimental system with eventing enhancements that are available with the new style. These principles will be applied from the experimental system to the final draft specification and API after more tests. If such a web-service standard meets the new binding possibilities, alerting will become widely accessible and GIS viewers and sensors can improve user-experience, loading/publishing sensor data or loading pipelined WMS tiles as well.

Keywords Sensor web · Web services · Web sockets · Performance · Complex event processing

1 Introduction

As the amount of sensor data is growing, more people want to see the data from the sensors online via the web. OGC SWE standards [1] enable the web-based discovery, exchange and processing of sensor observations, as well as the tasking of

J. Hübnerová (✉)
Faculty of Mechatronics, Informatics and Interdisciplinary Studies,
Institute of Novel Technologies and Applied Informatics, Technical University of Liberec, Studentská 2, 461 17 Liberec, Czech Republic
e-mail: jitka.hubnerova@tul.cz

© Springer International Publishing Switzerland 2015

I. Ivan et al. (eds.), *Geoinformatics for Intelligent Transportation*,
Lecture Notes in Geoinformation and Cartography,
DOI 10.1007/978-3-319-11463-7_9

sensors systems. SWE is technology to enable the implementation of Sensor Webs. Wildfires, river basins, tsunami alerts, and environmental risk management are just some of the uses of OGC's interoperability framework for web-based access and control of sensors and sensor data. One of the SWE standard's services, the relatively new Sensor Observation Service (SOS, 2008), provides an API (application programming interface) that allows web servers to collect data from subscribed sensors and public to explore their nearly real-time data. The goal of OGC Sensor Web Enablement SOS is to provide access to observations from plug and play sensors and sensor systems in a standard way that is consistent for all sensor systems including remote, in situ, fixed and mobile sensors. SOS standard is based on the REST (SOS 1.0) or SOAP (SOS 2.0) protocols. SOAP/REST protocols are the implementation of web services (and web services are a well-known application of SOA—service oriented architecture). Messages are exchanged using a request–reply pattern and interaction is synchronously initiated by client. The question is *if the standards are prepared today to be as interactive and interconnected to be usable from a global perspective*. Many sensor systems are built at a local level and their read-only data is published on the web. There is a large space for *linking these autonomous systems to the big sensor web and evaluating different event types with some higher automated logic or with preferences defined by each user*. As with other SOAP web services, performance may also become an issue and can negatively impact the user experience. Each request uses a shared HTTP persistent connection over a single TCP connection (in the best case) and waits for its response before another request can proceed. Web browsers open a memory-reasonable number of connections (for example 6 for Chrome) to partly overcome such limitations and developers use AJAX that prevents UI blockage (browser communicates synchronously). But the performance problem and other web service limitations still remains. At the time of writing, SOS standard are becoming known in web mapping software and first implementations exist (for example OpenLayers javascript library provides a very limited functionality for requesting the SOS service). So as we can see, for Sensor Observation Service, the mechanism is publicly available and open, which can overcome its other disadvantages. As for other OpenGIS standards, we think that this specification will become broadly popular in future. However, because of technology limitations, this web service stack cannot be used for real-time monitoring and alerting in particular. In the next few chapters we want to *describe our approach to overcome these limitations and also to present a version of an experimental system that we use to measure its performance parameters and to tune the specification draft*.

In 2003, Gartner introduced [2] a new terminology to describe a design paradigm based on events: Event-Driven Architecture (EDA). EDA [3, 4] defines a methodology for designing and implementing applications and systems in which events are transmitted between decoupled software components and services. Event objects are sent from an event source to the event consumer in asynchronous messages at times determined by the event source. Pushing event objects proactively reduces latency (the time required to respond to an event), compared to waiting for consumers to pull event objects (for example, by repeatedly asking if

any new data is available = polling). EDA had many forms during the years when it was used in local networks and now it is often discussed in relation to SOA and how these two can interact. This can be a very interesting feature when used in a World-Wide-Web environment for many uses and especially for sensor data publication and monitoring problems. Theoretical discipline (without practical application) which tries to combine these architectures is called SOA 2.0 (SOA 2.0 = SOA + EDA). Only local area network (LAN) monitoring and alerting systems are widely used today. One type of their output is sending SMS/e-mail messages to the specified group of users (e.g. crisis team) if some threshold is exceeded. This system is good for crisis team disaster early warning and is built with reliability in mind. *Such systems are not available to the public today*. For the public, another OGC Sensor Web Enablement standard was proposed and named Sensor Alert Service (and a very new Sensor Event Service). These SOA web service specifications have some disadvantages in transport binding which is firewall unfriendly (XMPP protocol for SAS) or requires the consumer to have a public endpoint address (SES). As we know, IPv4 is still the leading specification and not many users own such an address. From a global public monitoring and alerting perspective, these solutions are still weak for the task (and as a result they are not well-known).

The motivation for our work was dealing with performance issues of web services at first. After we built an API and experimental GIS-based system, new opportunities and topology enhancements were discovered. In the first part of the text we introduce some fundamentals of the proposed style to understand the concept and contribute with a comparison of the new concept with the style used today. Next we will describe the experimental system and publish the results of performance analysis. We would like to create another experimental system in future which should show a reduction in the time needed to load WMS tiles from the GIS server by pipelining enhancements of developed API (but that is not topic of this paper). This will show us how pipelining the capabilities of our architecture style can improve the performance of such a very common use. Finally we will describe topology and event processing enhancements which are interesting especially for “GIS on the web” use-cases and can be used for building a publicly available alerting solution (deployable to the cloud SaaS environment). The resulting description will be transformed to the draft specification and API after more tests on the experimental system.

2 Changing Architecture Style

The Weda architectural style is a hybrid architectural style that we have derived from other network-based standards, such as web services [5] and HTML5 websockets [6] to get a practical real-time SOA 2.0 [7] solution for WWW. It provides a uniform connector interface to the client and server implementers allowing them to extend their existing web services (SOAP 1.2, REST, POX) with a new type of

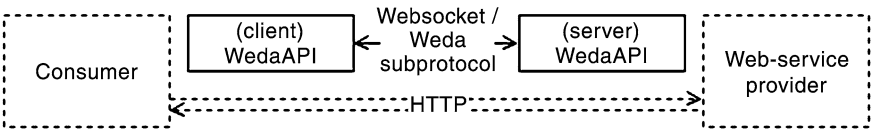

Fig. 1 Blackbox overview of Weda

endpoints and binding while keeping their HTTP server endpoints to legacy clients alongside Weda endpoints (Fig. 1).

New possibilities grew with the arrival of Websockets. Websockets is a technology that provides bi-directional, full duplex communication over a single TCP socket. It is designed to be implemented in web browsers and web servers and traverses firewalls, proxies, and routers seamlessly and leverages Cross-Origin Resource Sharing (CORS). The communication channel can be protected against eaves-dropping with TLS, much like HTTPS. The default ports are 80 or 443, so enterprises are not required to open additional ports in their firewalls.

2.1 Comparison of Web Services Architectural Styles

We can identify three classes of Web services (Table 1):

- REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations.
- RPC-compliant Web services, in which the service may expose an arbitrary set of operations.
- WEDA-compliant Web services, in which the service can use asynchronous message passing which can provide us eventing behaviour as well as call and return.

We developed an informal (IANA or RPC based) [8] as well as a formal (timed automata) specification [9], whose purpose is to ensure the interoperability between Weda implementers. The list of topics covered is: Weda gateway, Weda endpoints (also for non-public client endpoints), Addressing, Weda transport binding, Contracts, Weda subprotocol, Weda service description, model checking and verification. All of the components were implemented into the beta version of Weda API. The aim of future development is to provide an easily pluggable library in more programming languages, which has a simple interface but robust and self-contained implementation.

Table 1 Comparison of web services architectural styles

Attribute	WEDA-style	REST-style	RPC-style
Architecture	SOA 2.0	SOA	SOA
Distributed system type	Hybrid (message passing and call/return)	Call/return	Call/return
Addressability	Multiple endpoints per service (clients, server)	Unique URI address per resource	One endpoint per service
Common transport	HTML5 WebSockets	HTTP	HTTP
State	Statefull	Stateless	Stateless
Flow control	Asynchronous	Synchronous	Synchronous (over FW-friendly transport)
Process com. models	One-to-one, one-to-many, many-to-many	One-to-one	One-to-one
Latency	Best (after improving admission and flow control)	Good	Good
Throughput	Extremely high	Bad	Bad
Instance context	Per session	Per call	Per call
Scalability	Best in terms of concurrent clients	Good	Good
Coupling	Loose (only event type definitions in duplex contracts)	Functionally tightly coupled (MIME types in self-descriptive resource representations)	Functionally tightly coupled (operations and data types in contract)
Data interface	Inherited (no restriction)	Generic (e.g. HTTP verbs, MIME)	Service description (e.g. WSDL)
Common data format	Inherited (no restriction)	HTTP resource representation, XML, JSON	SOAP
Deployment topologies	Enterprise service bus	Hub and spoke (centralized)	Hub and spoke (centralized)
Coordination	Esb's native functions for orchestration and choreography, no scheduler	Resource-oriented workflows (theoretical-atom, rss, dynamic hyperlinks in practice)	Service-oriented workflows, scheduler required
Coordination	Esb's native functions for orchestration and choreography, no scheduler	Resource-oriented workflows (theoretical-atom, rss, dynamic hyperlinks in practice)	Service-oriented workflows, scheduler required

3 Experimental System

We have built two experimental "GIS on the web" systems with WEDA API. Experience and data obtained from these experiments were used for calibrating the model. Both experimental systems use the same server-side implementation and only client implementations differ as one was developed as a thick client and the other as a thin client (Fig. 2).

3.1 Server Side

The server side consists of a database layer, data access layer, web-service and server-side Weda API.

- *Database layer*—Sensor data is stored in the spatial database Observations Data Model (ODM). Version 1.1 [10] is a generic template for the observations DB. For example the SpatialReferences table provides specifications of the location of an observation site to record the name and EPSG code of each spatial reference system used. The database was running inside a MSSQL 2008 environment. We used more types of data, for example hydrologic data from CUAHSI-HIS.
- *Data access layer*—Our data access layer provides us mapping of conceptual schema to data schema, isolation from the relational database and database schema and other features.

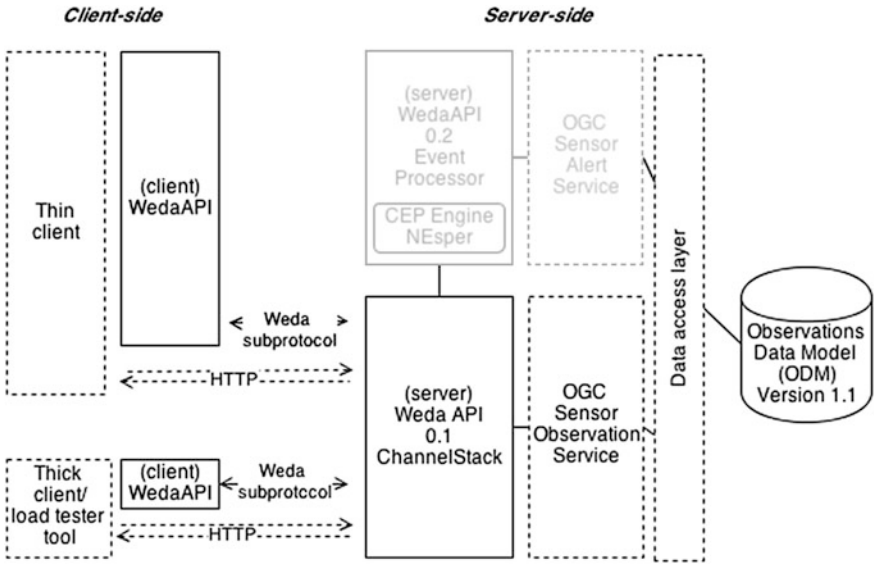

Fig. 2 Overview of experimental systems

- *Web service*—As we wanted to be sure that the existing service could be extended, we chose OGC Sensor Observation Service [11] as part of our experimental system. The server solution consists of the implementation of standard SOS webservice without changes in contracts and business logic (the goal). For spatial data, Renci (Renaissance Computing Institute) OpenGIS implementation was used to bring us API for using Gml, Ows, SensorML or Tml specifications. In the first version of samples we use its Core and Enhanced extensions with GetCapabilities, DescribeSensor, GetObservation and GetFeatureOfInterest operations. In the future version transactional extension of SWE SOS can be implemented especially with a proposal of one-way InsertObservation operation and broadcast event that new observation arrived for all clients.
- *Weda API*—Other projects undergoing development are WedaAPI and layers of the Weda eventing processor (will be integrated to WedaAPI after more tests). The WebSocket server used is RFC6455 Super-WebSocket implementation. The Weda eventing processor integrates a Complex event processing engine NEsper—the widely used CEP engine offering runtime for .Net. CEP server runs independently and has its own long running lifetime over the requests. REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations.
- RPC-compliant Web services, in which the service may expose an arbitrary set of operations.
- WEDA-compliant Web services, in which the service can use asynchronous message passing which can provide us eventing behaviour as well as call and return.

3.2 Thin Client

Figure 3 shows the web client interface of our thin-client connected to the server. Both clients read the geo-spatial data from the OGC SOS service by Weda ChannelStack—transport and message binding and subprotocol. This client acts as GIS Web map reader with WMS and SOS layers. It is implemented with ASP.NET MVC3 and JavaScript using OpenLayers. We extended its Protocol.SOS javascript library to be capable of connecting to the Weda endpoint. The use for the client is as a public GIS Viewer system which presents SOS service data graphically upon the public WMS layer while that data is loaded over Weda. End developers can build a nice viewer with many features according to Weda capabilities. This client was not considered as a benchmarking environment. Nevertheless some response time logging is contained in source so the user can optimize the application after displaying the response time information in the browser's console.

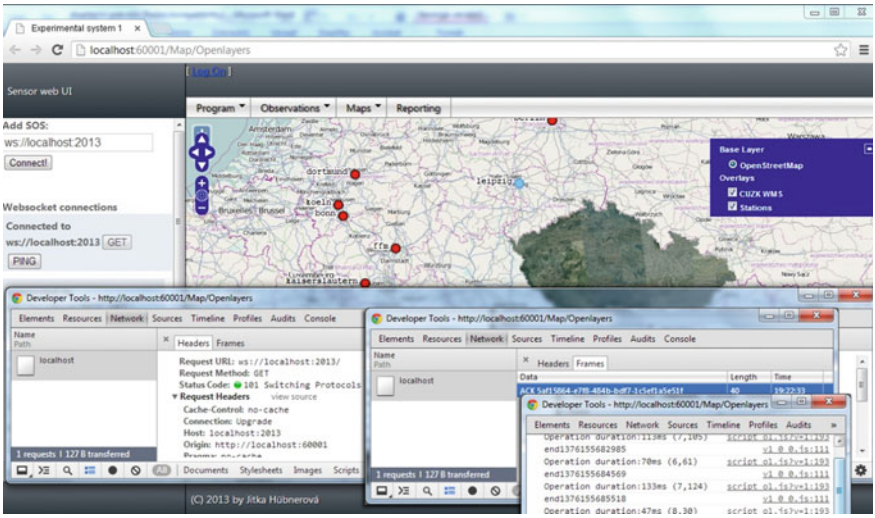

Fig. 3 Experimental system with thin client interface

3.3 Thick Client

Figure 4 shows the desktop client interface of our thick-client. It is implemented in C#.Net Winforms. The desktop client application was extended to be a load testing tool. As Websocket is a new protocol, there are no load testing tools that can act over WebSocket and none extensible with some subprotocols. This client allows us to do real benchmarks of Weda against REST and SOAP over HTTP SOS service. Legacy SOAP/REST endpoints are also invoked and used in benchmarks as baseline.

4 Performance Analysis

We measured response time instability of Weda by invoking number of requests (according to SOS GetCapabilities, DescribeSensor, GetObservation and GetFeatureOfInterest operations) from the thick client application and collecting the responses with metadata about server processing times and other parameters (such as 20 kB amount of transferred data per request etc.).

The load generator was hosted on 4xIntel Xeon running at 2.5 Ghz, Windows 8, 2 GB of RAM, 1Mbps downlink network connection and 100 Kbps uplink network connection. The location of the load generator was 4 network hops away from the server hosting the service. Reverse proxy (no caching) was placed between the client and server. The average packet round trip time was 33 ms and constituted less

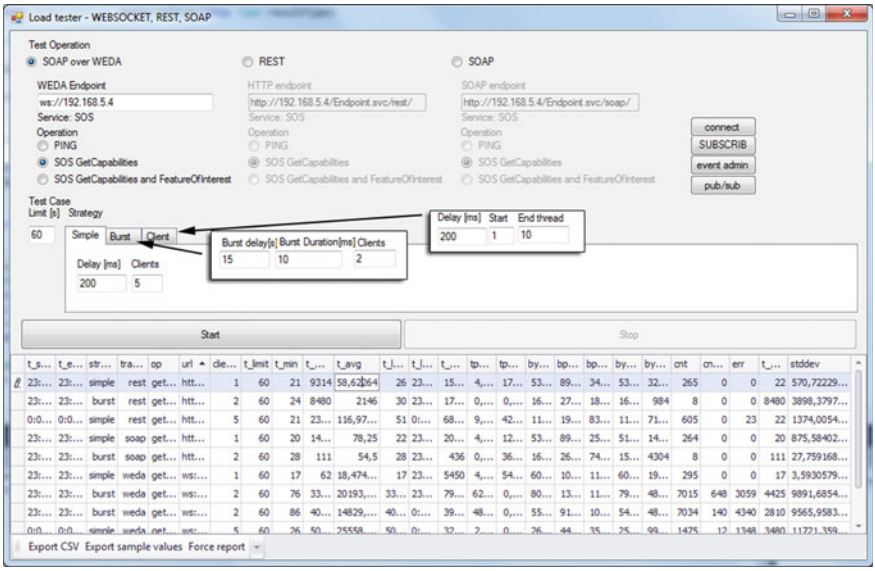

Fig. 4 Experimental system with thick client interface

than 1 % of the service time. The server was hosted on an Intel Core i7 2670qm running at 2.2 Ghz, 4GB DDR3 665 MHz, Windows 7 professional sp1 64bit. The database server (MSSQL 2008 R2) was running on the same host as the Weda server so its latency is included in total amount of RPT. It was found that RPT times mainly consist of latency of data access layer (99 %). The test case for measuring response time instability has been defined with constant payload of GetCapabilities operation invoked at OGC SOS webservice. Every 10 s for 3 h a request was sent and results were measured to give us more than 1,000 samples. The server processed each request by proper serialization at each layer up to the bottom data access layer. Backward propagation of results was packed into response frames by Weda API and metadata about server processing times was glued into the response. An illustration of setup and measuring points can be seen in Fig. 5.

RTT includes a time for request forwarding achieved by our reverse proxy. This was used to simulate such a device's delay. The Weda architectural style is a hybrid architectural style that we have derived from other network-based standards, such as web services [5] and HTML5 web-sockets [6] to get a practical real-time SOA 2.0 [7].

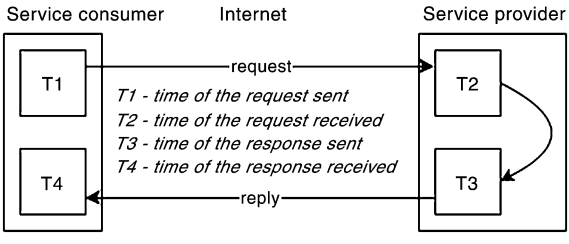

Fig. 5 Benchmark setup

$$RT = T4 - T1 \tag{1}$$

$$RPT = T3 - T2 \tag{2}$$

$$RTT = (T2 - T1) + (T4 - T3) = (T4 - T1) - (T3 - T2) = RT - RPT \tag{3}$$

Performance trends and variance results are shown in Fig. 6. It can be seen that response times are constant for 50 % (1,600–1,700 ms) samples. 73 % of the sample's RT were around the 95th percentile. 26 % of samples have uncertain response time varying from 2 to 11 s (four times more than average value). RTT is the main part of RT value. Its distribution is very similar to RT as 48 % of RTTs are between 1,600 and 1,700 ms. One small peak can be found at 4.5 s where 7 % of samples are situated.

Table 2 shows the statistics of the test. A ratio between standard deviation and average value is used as an uncertainty measure. From the table we can see that RT and RTT have a relatively small variance but RPT has significant instability which does not affect the final RT by much.

A very small amount of samples are significantly affected by RPT giving more than 1 s to total RT. From this point there is no chance to significantly improve performance by improving serialization technique (except adding the compression) or dealing much with the implementation. In our other work we use these results to predict response time instability formula.

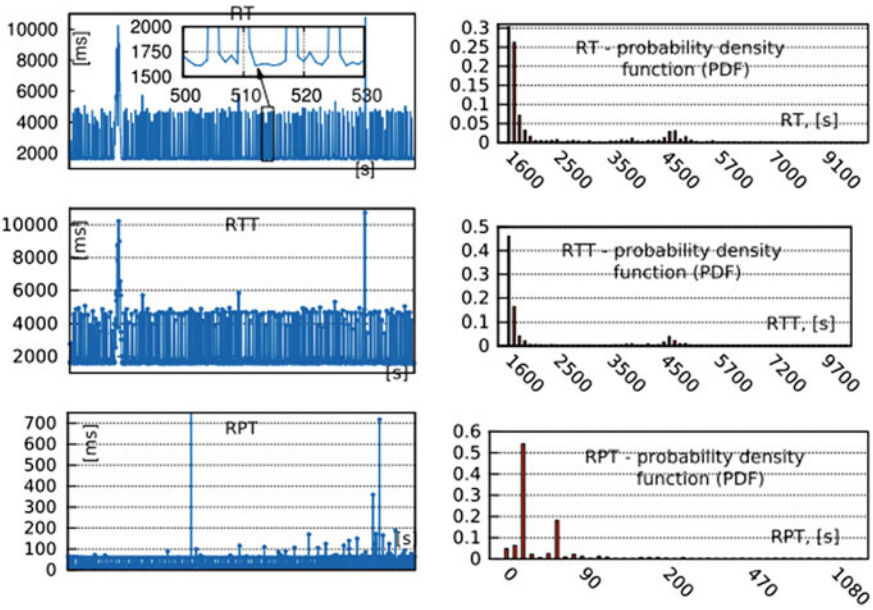

Fig. 6 Performance trends and probability density of RT, RTT, RPT

Table 2 Performance statistics: RT, RPT, RTT

	Min (ms)	Max (ms)	Avg (ms)	95th (ms)	Std.dev.	Std.dev/ Avg (%)
RPT	10	1,280	31	18	54	940
RTT	1,580	10,754	2,197	1,630	1,244	56.6
RT	1,608	10,773	2,258	1,669	1,243	55.8
Ping RTT	32	367	40	35	3	1.7

4.1 Baseline Benchmarks

We performed many benchmark measurements to compare Weda against REST and SOAP/RPC web services. These results and comparative graphs are out of the scope of this paper, so here I only wish to add some findings on interesting quality attributes.

- *Findings on the throughput attribute*—from "burst-based test cases" we can learn that synchronous styles (SOAP over HTTP and REST) can only achieve a small amount of turns compared to Weda-style. Weda-style has a 40-times higher throughput, but as such it is more susceptible to DDoS attacks without a robust admission control mechanism (tests ran without any admission control mechanism implemented). A great way of dealing with overwhelming issues is to add an admission control mechanism at each input queue. It is a matter for discussion if such a mechanism should be required directly in WebSocket specification (not in Weda-style).
- *Findings on the scalability attribute*—very interesting results were obtained from the "constant count of samples per burst test case," which suppressed the differences caused by asynchronous or synchronous transport. We saw that throughput increases exponentially with the number of clients for Weda-style. RPC and REST-styles have their peak-throughput relatively low at a count of 6 clients (each client invoked exactly 10 samples per burst every 1 s). Weda-style proves that it is more scalable in terms of concurrent clients.
- *Findings on the response time attribute*—peak-throughputs can negatively impact Weda-style (the next version should deal with it with an admission/flow control mechanism). To suppress this behaviour we prepared a test case where conditions were set in a way leading to very similar throughput behaviour (we prevented Weda-style to send/process more samples than other styles). From the results we obtained that Weda's 90th percentile response time is lowest and unaffected by incrementing client count unlike the RPC and REST-style. This test case shows that Weda-style responsiveness is a little bit better than for RPC and REST-style.

5 Event Processing Enhancements

The main building block of web based GIS monitoring and alerting solutions is contained inside the Weda specification—the use of duplex services. This enables message exchange patterns in which both endpoints can send messages to the other independently. A duplex service, therefore, can send messages back to the client endpoint, providing event-like behaviour. Duplex communication occurs when a client connects to a service and provides the service with a channel at which the service can send messages back to the client. We can benefit from the client's Weda endpoint which is accessible from the server. To implement the push mechanism, the client must implement a client-specific contract called a callback contract. As we created our experimental system before the SES standard was proposed, our experiments contain an easier WS-Eventing [12] contract (other WS-Notification [13] OASIS-Standard is bind able to the model). There are three types of services needed in enhancements:

1. Subscription and notification management
2. Default public integration point for sensors, monitoring systems and other event sources
3. Integration point for admin tools for statement/topic management.

As shown in Fig. 7, the Weda event processor consists of a dispatcher component, four event processing services which can be running on separate instances and one CEP engine. Complex event processing is technology to transform single, low-level events into aggregated, high-level events by looking across event streams. Many message types are transmitted here as SOAP management operations, events, subscription messages, registered EPL rules and rule actions. Implementation of eventing enhancements is now integrated in an experimental system only. After

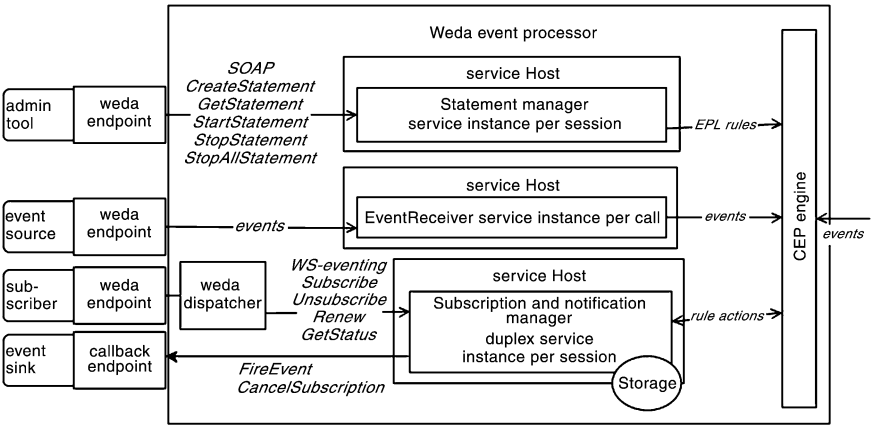

Fig. 7 High-level process view of Weda event processor and its relationship to EDA components

stabilization of API and testing with SES specification, it will be integrated directly into WEDA API. In the scope of this paper we will only highlight some of the components that build together the practical implementation of SOA 2.0 architecture.

- For example the Notification manager component reacts on rule actions from the CEP server and parses the list of subscriber's topics to make the correct push of an event to appropriate event sink. Events are defined by the end application. An example of event-type (SASAlert) follows:

```
<?xml version="1.0" encoding="UTF-8"?>
  <SASAlert xmlns="http://www.opengis.net/sas"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><Header>
        <AlertMessageStructure>
          <sas:QuantityProperty>
             <sas:Content definition="urn:ogc:def:property:OGC:Temperature"
   uom="Cel">-0.9</sas:Content>
          </sas:QuantityProperty>
        </AlertMessageStructure>
      </Header>
      <Body>51.96 7.607 70.0 2009-10-17T02:27:04Z 0 30</Body>
  </SASAlert>
```

- Event generator integration point—EventReceiver metadata service is the main integration point for monitoring systems, network sensors and other event generators (event sources) which send an event into the CEP server for further processing.
- Statement manager metadata service is an integration point for any administration tool that allows definition of topics. Thanks to the StatementManager service, experts can provide a set of rules that may change over time, due to the dynamic nature of the domain. Client application behaviour can be changed only by changing the set of rules, nothing has to be programmed. An example rule provided as EPL statement follows. This example statement fires as soon as a LocationSensor of a certain device does not fire events for 10 s. Every user can then subscribe for this topic.

```
SELECT count(*), Identifier FROM LocationSensor.win:
time(10 s)
GROUP BY LocationSensor.Identifier HAVING count(*) = 0
```

6 Conclusions

In this work, the author presents the application (experimental system) of the WEDA architectural style for developing "GIS on the web" solutions. The paper also shows performance results measured on the system. It shows that RT is stable

and good enough to be used in real-time and also summarizes other benchmark findings from a number of different test cases. The system can improve the performance of sensor web services and thanks to the presented "event processing enhancements" it extends messaging capabilities for publicly available monitoring and alerting sensor webs. There are many applications for this system from small ones (e.g. warning the public or farmers within a range of 10 km before an approaching storm or hail) to bigger ones (monitoring of emissions in real time by informing the public to close windows and alerting the relevant authorities after exceeding permitted limits) or building automata that can warn before some critical event occurs (if water exceeds the threshold at Liblín and Zvíkovec and it is raining in Beroun, then clients are alerted from Beroun to Praha 11–13). With this technology, results from very different sensor types can be processed together and cross calculated. Users can define their own preferences for what to monitor and alert for. The resulting trend analysis can be made available on a global level and deployed in the cloud environment.

Acknowledgments This work was supported by the Ministry of Education of the Czech Republic within the SGS project no. 78001/115 on the Technical University of Liberec.

References

1. Botts M, Percivall G, Reed C, Davidson J (2008) OGC sensor web enablement: overview and high level architecture. In: Nittel S, Labrinidis A, Stefanidis A (eds) GeoSensor networks, volume 4540 of lecture notes in computer science, Berlin, 17–20 Sept 2007, Springer, pp 175–190
2. COM-20-2737 (2003) Event-driven applications: definition and taxonomy. Gartner, Stamford
3. Etzion O, Chandy M, Ammon RV, Schulte R (2006) Event-driven architectures and complex event processing. IEEE international conference on services computing, IEEE, Sept 2006
4. Chandy KM, Charpentier M, Capponi A (2007) Towards a theory of events. In: Proceedings of the 2007 inaugural international conference on distributed event-based systems, DEBS '07, ACM, New York, USA, pp 180–187
5. Group WW (2013) Web services glossary (2004) Available via DIALOG. http://www.w3.org/TR/ws-gloss/. Accessed Oct 2013
6. RFC6455 (2011) The WebSocket protocol. IETF, California
7. Levina O, Stantchev V (2009) Realizing event-driven SOA. In: Perry M, Sasaki H, Ehmann M, Bellot G, Dini O (eds) Fourth international conference on Internet and Web applications and services, Venice/Mestre, Italy, 24–28 May 2009, IEEE Computer Society, Washington, pp 37–42
8. Hübnerová J (2013) Weda—new architectural style for world-wide-web architecture. In: Proceedings of ISAT 2013, 34th international conference information systems architecture and technology, Sept 2013, Poland, Institute of Informatics Wrocław University of Technology, Poland, ISBN 978-83-7493-804-4
9. Hübnerová J (2013) Model based analysis and formal verification of WEDA architectural style. In: Proceedings of IEEE international conference on informatics and applications (ICIA), Poland, Sept 2013, Poland, IEEE, Poland, ISBN 978-1-4673-5255-0
10. Tarboton D, Jeffery S, Horsburgh TD, Maidmen JS (2013) Cuahsi community observations data model (ODM), version 1.1, design specifications

11. OGC 06-009r6 (2008) Sensor observation service v.1.0.0. OpenGIS implementation standard, Open geospatial consortium
12. Box D, Cabrera LF et al (2004) Web services eventing (WS-eventing) specification. Available via DIALOG. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-eventing/WS-Eventing.pdf. Accessed Oct 2013
13. Graham S et al (2006) Web services notification (WS-notification) version 1.0. Available via DIALOG. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn#technical. Accessed Oct 2013
14. Clements P, Kazman R, Klein M (2001) Evaluating software architectures: methods and case studies. Addison-Wesley, Boston