Measurement, Monitoring and Environmental Assessment research program (MMEA)
Research report
Deliverable 1.1.7

Helsinki 2013

Extensible Architecture for Distributed Near Real-Time Environmental Monitoring

Authors

Corresponding author: Ville Kotovirta [VTT Technical Research Centre of Finland]
P.O. Box 1000, FI-02044 VTT, Finland.
Tel. +358-20-722 111, fax. +358 20 722 7024
e-mail: ville.kotovirta@vtt.fi

2nd author: Harri Hytönen [Vaisala Oyj]
P.O. Box 26, FI-00421 Helsinki, Finland
Tel. +358-9-894 91
e-mail: harri.hytonen@vaisala.com

3rd author: Antti Aalto [HiQ International AB]
Vaisalantie 6, 02130 Espoo, Finland,
Tel. +358-9-4355 860
e-mail: antti.aalto@hiq.fi

CLEEN LTD
ETELÄRANTA 10
P.O. BOX 10
FI-00130 HELSINKI
FINLAND
www.cleen.fi

Abstract

Near real-time environmental monitoring is needed for decision-making in dynamic environmental conditions. A distributed near real-time environmental monitoring system consists of distributed processing components that produce information from the raw data coming from distributed sources. The data processing needs to deliver the information in time, be reliable and robust to succeed in the delivery continuously, and be cost-efficient to develop, configure and maintain. We discuss 1) requirements for a distributed near real-time environmental monitoring system; 2) suggest an architectural model that supports the development of scalable, extensible, interoperable and robust near real-time environmental monitoring systems; and 3) present a prototype system implementation for validating the architectural model. The architecture combines Enterprise Service Bus (ESB), complex event processing (CEP) and cloud computing, and the prototype set up can do simple event processing for 7,000 observation messages per second with processing costs of around $2/hour.

Keywords: Environmental monitoring; near real-time; data processing; cloud computing; Enterprise Service Bus; complex event processing; distributed computing

Contents

1 Introduction

Large amounts of environmental data are produced by various distributed sensor networks and forecast models in near real-time. These data are the basis for near real-time environmental monitoring systems that process relevant context-related information from raw environmental data and deliver the processed information to end-users who utilize it in decision-making in dynamic environmental conditions. For example, a system for monitoring ice conditions in ice-infested sea areas delivers satellite and on-ground radar images, as well as oceanographic and meteorological observations and forecasts, to ships for navigation purposes (Kotovirta et al., 2011); an air quality monitoring system collects air quality measurements from distributed sensor networks and generates visualizations as well as sending alerts to citizens about the changing air quality (Lim et al., 2012); a tsunami early warning system collects seismic observations, buoy observations and tide gauge observation about water level from various areas to produce customized warning messages for delivery via different channels such as the web, TV broadcasting, SMS and e-mail (Wächter et al., 2012).

1.1. Near real-time processing

The term *near real-time* here refers to monitoring systems that operate in approximately second or minute scale depending on the application-specific requirements. Changes in the environment are detected and processed without unnecessary delays, but the requirements are not as high as for the *real-time* monitoring systems applied for example in automation and robotics that operate in millisecond scale. However, for decision-making in dynamic conditions the information should be delivered in time, as it becomes outdated when the conditions change. For example, in the ice conditions monitoring example the latest satellite image is delivered to the icebreakers in tens of minutes after the satellite flyby, which is adequate for navigation purposes. The information delivery should be successful in all cases, as a failure to deliver critical information may be costly, even life-threatening. For example, ships navigating in ice are dependent on updated information about the ice conditions in order to choose safe and cost-efficient routes. In the case of a near real-time air quality application, allergic people must remain aware of surrounding air quality in order to take their medicine in time, and people living close to a factory where a chemical release has just taken place must be alerted about the accident and advised to take cover.

1.2. Distributed systems

A near real-time environmental monitoring system can be considered *distributed* when the data sources, the data processing components and the end-user applications are distributed among heterogeneous systems and organizations. The raw data sources may include various in-situ or remote sensors and also forecasting models interpolating and extrapolating the measurements in space and time. For example, the ice conditions monitoring system (Kotovirta, 2011) collects oceanographic and meteorological observations and forecasts from various met offices and environment institutes, radar images and ice observations from the ships, and remote sensing data from private and public satellite data sources, and delivers the processed information over a mobile link to icebreakers operating in the Baltic Sea.

There will be more and more distributed near real-time data available for environmental monitoring, as regional and global initiatives are aiming at integration of environmental systems. For example, Open Geospatial Consortium (OGC) aims at interoperable interfaces that enable sharing of environmental data, and Global Earth Observation System of Systems, GEOSS, is a world-wide voluntary effort aiming at global connectivity of already existing systems monitoring the environment and storing environmental data (GEO, 2010). Current trends are towards opening all publicly produced data, including environmental data. In Europe the INSPIRE directive (Infrastructure for Spatial Information in the European Community) obliges European public organizations to open up their environmental data sources for applications (European Parliament, Council, 2007).

1.3. The scope of the article

In this article we consider the development of a system to support the development of near real-time environmental monitoring systems that utilize data coming from various distributed data sources. Although many sources are available with standard interfaces, the development of robust and cost-efficient solution to answer end-user needs requires careful engineering. Data sources need to be found, data semantics understood, interfaces adapted, processing tasks developed, and the relevant information processed, delivered and presented to end-users. In addition, the development and runtime costs need to be taken into account.

We analyze general requirements for distributed near real-time environmental monitoring, and suggest an architectural model that enables development of systems fulfilling the requirements. The architecture is based on state-of-the-art architectural and information technology components such as Enterprise Service Bus (ESB), complex event processing (CEP) and cloud computing. We also present a prototype implementation of the architecture and its validation results.

2 System architecture

2.1. *Requirements for distributed near real-time environmental monitoring*

In this chapter we analyze the general requirements for a distributed near real-time environmental monitoring system. Figure 1 shows a reference model of such a system. The data sources include raw measurement data collected by sensors, as well as the data extrapolated and interpolated in space and time by models. The data from the sensors and models are processed by various processing components (e.g. filtering, data fusion) and personalized to various end-users, humans using a variety of devices, or computers requesting a variety of interface protocols. Personalization includes the end-user application or software that represents the information to the user. The monitoring system consists of data processing chains controlled by a processing chain control module that monitors and manages the operations of the chain. Historical values and processed results can be cached or stored for later use. The processing and personalizing components are separated to emphasize the difference between the production of more “bulk-like” data (i.e. preprocessing) and the personalization of the bulk data for specific end-user needs and user context, such as location, user role, user devices, software and interfaces. The data processing chains, i.e. the whole environmental monitoring system, reduce the amount of data, and increase the amount of information, i.e. usefulness of the data, from the end-user point of view.

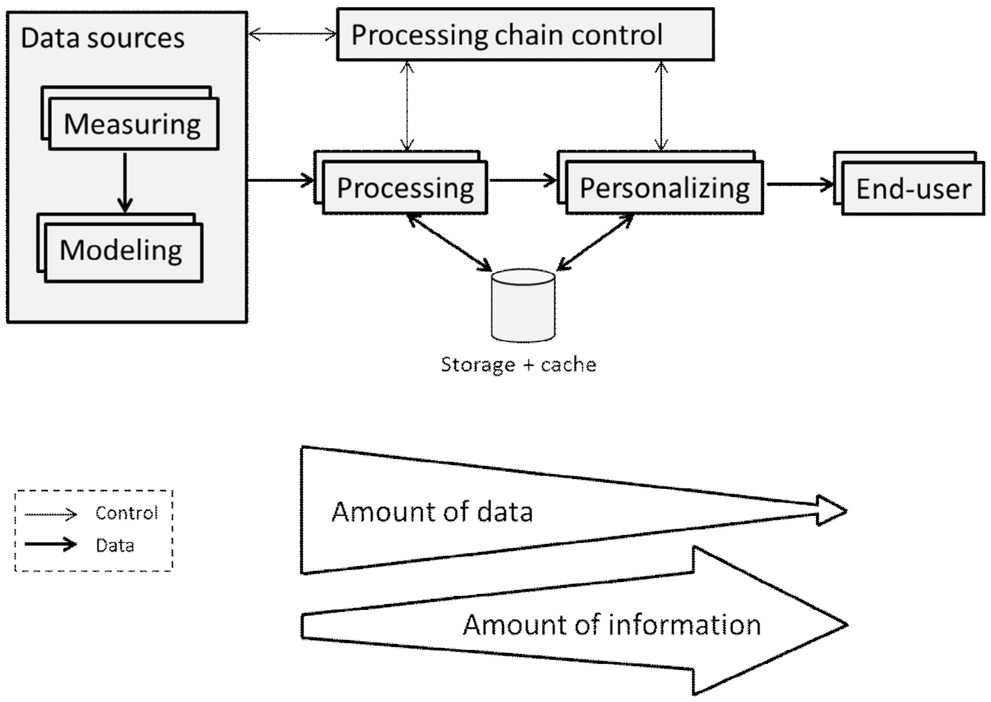

Figure 1. A reference model of a distributed near real-time environmental monitoring system. Data coming from sensors and models are processed and personalized to end-users so that the amount of data volume is minimized and the amount of information is maximized.

When designing and developing state-of-the-art near real-time environmental monitoring systems, the following issues, or requirements, should be taken into account. It should be noted that these are general requirements specified by the application under development. The requirements are:

Integration of various distributed systems. Environmental data are produced by many sensors and sensor networks, including in-situ sensors (fixed, ad hoc, wired, wireless), people using their mobile phones remote-sensing sensors such as radars and satellites, and computer models producing forecasts.

Extensibility. The monitored environment is unpredictable, and data input and output as well as data processing needs may grow as a result of a sudden event in the environment. The system should be extensible so that new data, processing and users can be included.

Configurability. New uses of the system may be achieved, not by adding new data sources or processing components to the system, but by re-configuring the system's existing components. Also, the system's performance may be adjusted or optimized through configuration – of data computing resources, data storage capacity, end-user roles, or security policy, for example.

Scalability. When new data sources, processing and end-users are introduced in the system more computing power, data storage and data transmission capacity is required.

Timely data delivery. The system should deliver information in time, i.e. the resources reserved for data processing, data storage and data transmission should be adequate for the time requirements of the application area and the end-users.

Relevance of data delivery. The data delivered to the end-user should contain only the relevant information, i.e. the data are personalized to the user needs.

Operational reliability. The system should be successful in delivering the information in every case, as a failure to deliver critical information may be costly, even life-threatening. In addition, there should not be false alarms as they will reduce user trust in the system.

Data quality / reliability. Data quality becomes a relevant issue when important decisions are made based on the processed information. The quality of the information produced by the data processing chain is dependent on the data quality not only of the raw data sources, but also of the computational components of the processing chain.

Data storage. Although the system considers near real-time data flows, historical values may have a significant role in the decision-making. Examples of earlier situations may help in estimating how the conditions develop, and change detection algorithms may compare the newly measured values with the older ones. If the data sources do not provide historical values, and the values would give additional value to the data processing, the monitoring system should store them and enable efficient data retrieval.

Data security. The more valuable the information produced by the system, the more important the data security of the data processing chain. The key principles of data security must be included, i.e. confidentiality, integrity and availability.

Cost efficiency. In principle the value gained from a monitoring system defines how much the system can cost. Some critical applications must be operating reliably with strict data security regardless of the costs, but in many cases costs are restricting the architectural choices of the system and thus defining its overall performance and usability.

2.2. System architecture for developing distributed near real-time environmental monitoring systems

In this chapter we describe the suggested system architecture for developing distributed near real-time environmental monitoring systems fulfilling the requirements defined in chapter 2.1. The architecture is based on the Enterprise Service Bus (ESB) software architectural model and cloud computing platforms, and is presented in Figure 2.

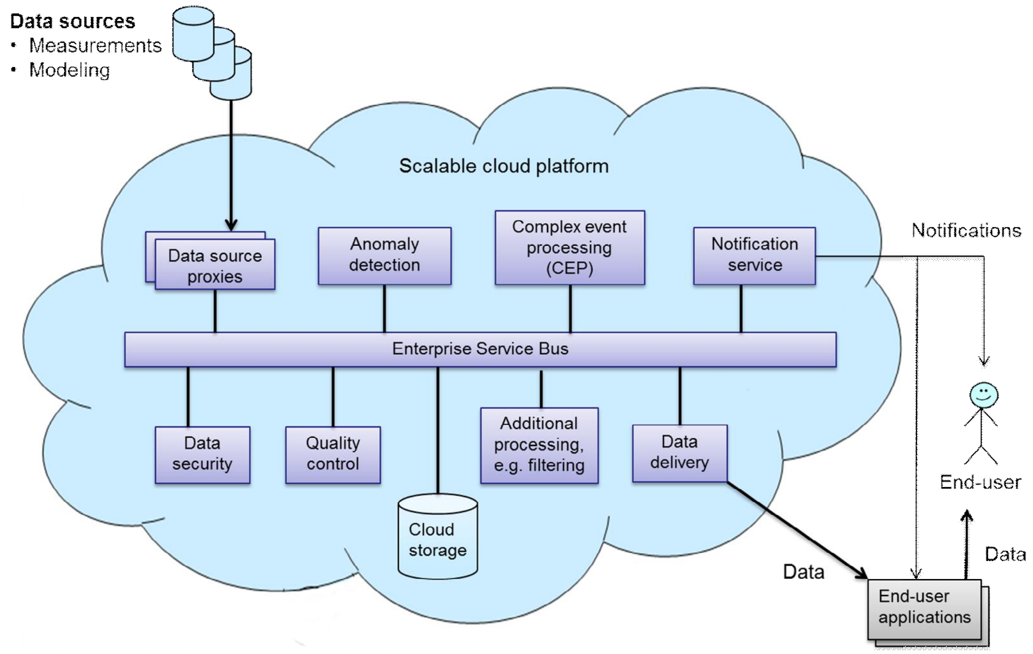

Figure 2. The system architectural model for distributed near real-time environmental monitoring systems. An Enterprise Service Bus mediates messages between various services, and controls the data processing chain. The ESB and processing services run on a scalable cloud platform, and cloud storages are used for data storing and caching.

The requirement for *integrating various distributed systems,* such as data sources and data processing components, suggests the use of the Service-Oriented Architecture (SOA) model which is ideal for integrating heterogeneous systems (Erl, 2005). In this model, the data sources and processing components provide each other with services, i.e. interface procedure calls that can be accessed by other components, thus enabling creation of a distributed data processing chain. However, the plain SOA model does not yet answer the *operational reliability* requirement and provide robust data processing chains that connect various data sources and processing modules together. The data processing chain needs to be controlled.

The Enterprise Service Bus (ESB) is a component that mediates messages between various distributed systems by transforming and routing the messages (Chappell, 2004). It extends the traditional client-server model and promotes asynchronous message oriented design for communication between systems. The ESB implements a central system that controls and tracks the message delivery, thus enabling more robust performance of the data processing chain. For example, if some processing component fails to complete its task in time, the central control can trigger other service calls that try to solve the problem, go around it, or alert a human operator. The ESB supports service virtualization, i.e. the components in the processing chain are virtualized in the workflow control, and the components can therefore be substituted with other components providing similar functionality. This enables flexibility and the required *extensibility* in the data processing, thus improving the robustness, *operational reliability*, *configurability* and customizability of the processing task. For example, if some input data source fails to deliver data, the source could be substituted with another source providing similar data. In addition, if some processing module ceases to function, the system could use an alternative module to finalize the calculations in time.

To improve the *operational reliability* of the processing chain, the control module and the processing components of the chain are operating on reliable, scalable and easily manageable cloud platforms provided by the emerging field of cloud computing. Cloud computing is foreseen to provide the 5th utility, after water, electricity, gas and telephony, for meeting the everyday computing needs of the general community (Buyya, 2009). E.g. Lee et al. (2010) demonstrated that cloud computing resources are sufficiently elastic to deal with the unpredictable loads of real-world sensing applications. With cloud services it is *cost-efficient* to set up computing systems with required data processing, *data storage,* networking and uptime availability without the need to invest in and maintain the computing hardware and required expertise.

It is also cost-efficient to adjust resources to meet the changing needs for data storage and processing, thus taking care of the *scalability requirement*. As the services are virtualized, each service can run on a separate node or a cluster of nodes in the cloud. The easier adjustment of resources also helps in implementing the required update frequency, as more processing power is available if *timely delivery* and timely data processing asks for it. On the other hand, the processing can be scaled down and related costs reduced, if less processing is required. The data can be automatically replicated to different server farms in different geographical locations, depending on the cloud service provider, which improves the *reliability* of *data storing*.

A dedicated component in the architecture is responsible for the *quality control* of the dataflow. Quality processing can range from simple threshold checks to more complex statistical analysis, such as the one presented by Hill et al. (2010). Any deviations in expected data quality are detected, and application-dependent actions are taken, e.g. augmenting metadata, diverting the erroneous dataflow to another processing chain, generating an event, and notifying and alerting end-users, maintenance, or other relevant stake-holders.

To fulfill the requirement of delivering only relevant information and reducing the amount of delivered data (*relevance of data delivery*), the architecture contains an anomaly detection component, a complex event processing (CEP) component and a notification service. When the quality control is needed for detecting quality problems in the data, the anomaly detection is for recognizing relevant or interesting features in the data and in the environment that the data represent. For instance, a decline in the forecasted air quality, or high winds caused by an approaching storm center, are examples of anomalies in the environment that the user might be interested in. The general architecture does not specify any anomaly detection algorithm, as the solution depends on the application. Algorithms for near real-time anomaly detection of sensor data have been developed, e.g. by Yao et al. (2010).

The detected anomalies are represented as events for the complex event processing (CEP) component which can combine events from multiple anomalies to infer patterns of more complicated events. CEP consists of a real-time event correlation mechanism controlled by rules encoded with an event pattern language (Luckham, 2002). The events are the basis of notifications about meaningful changes in the environment that are delivered to end-users or end-user applications by the notification service. As a result, users do not necessarily need to retrieve or poll data regularly, but are notified whenever something relevant happens from their point of view. Users do not have to be active, but the system actively pushes relevant data to them, thus enabling a need-based data delivery which reduces the total amount of data transferred. After a notification, the user or the end-user application can retrieve more data describing the situation. The data may be processed, e.g. filtered, by the data processing chain to better answer the end-user or end-user application needs. The data delivery component is responsible for delivering the data to the end-user application which presents the data to the end-user. The data delivery is triggered either by events or by user queries.

The data processing chain's *data security* is highly dependent on the level of data security of separated services. However, as the Enterprise Service Bus (ESB) acts as a central point of communication it can improve the overall data security by controlling the traffic between the services. The data security service component in the architecture takes care of message confidentiality and integrity, identity and authentication, authorization and privacy, access policies, and federation of identities. The level of data security is adjusted according to the specific requirements of the application.

3 Prototype system for validating the architectural model

In this chapter we present the results of validating the architectural model by a prototype implementation. The prototype was implemented on commercially available cloud computing platforms using open source and proprietary components, and its performance for different requirements was evaluated. In most cases the performance with relation to the general requirements could be evaluated only qualitatively, as quantitative measures were not defined or set by application-specific requirements. However, some quantitative values could be measured, such as those for performance and scalability.

3.1. Prototype architecture

The prototype architecture is presented in Figure 3. The backbone of the prototype is based on the WSO2 Enterprise Service Bus (ESB) open source implementation for the service virtualization, and mediation of messages between services. The prototype was installed on Linux nodes that run on the commercially available Amazon Elastic Compute Cloud (Amazon EC2), and used Microsoft Azure cloud storage and PostGIS SQL database for data storing.

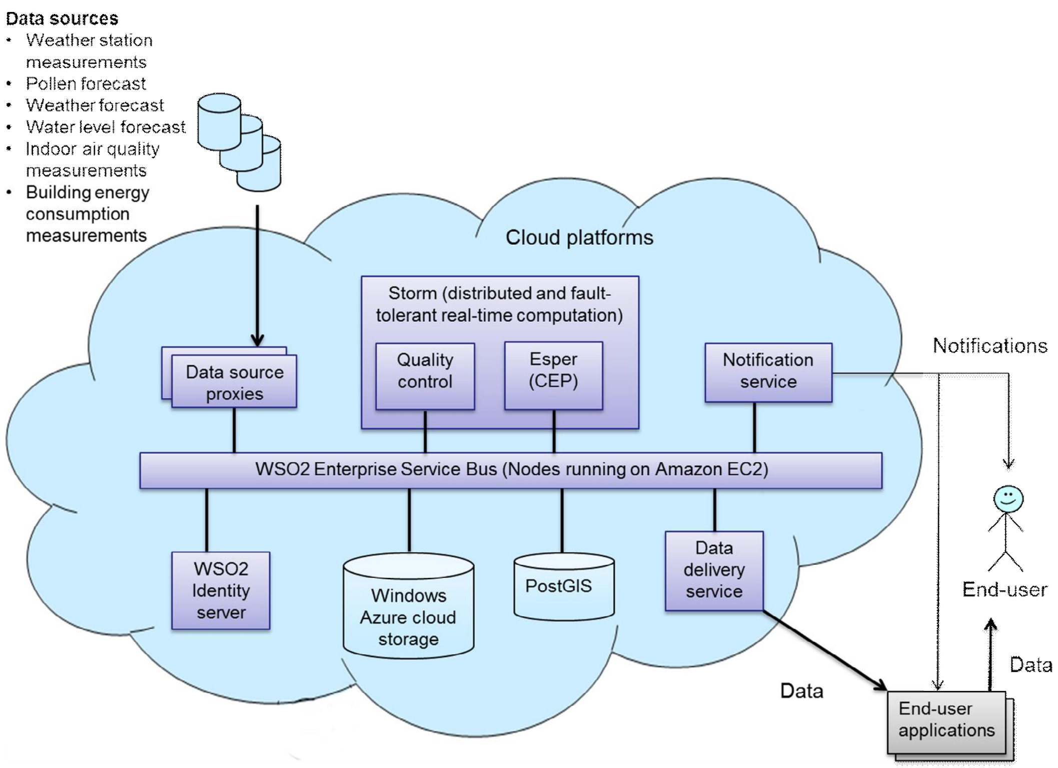
Data sources
• Weather station measurements
• Pollen forecast
• Weather forecast
• Water level forecast
• Indoor air quality measurements
• Building energy consumption measurements
Cloud platforms
Storm (distributed and fault-tolerant real-time computation)
Quality control
Esper (CEP)
Notification service
Notifications
Data source proxies
WSO2 Enterprise Service Bus (Nodes running on Amazon EC2)
WSO2 Identity server
Windows Azure cloud storage
PostGIS
Data delivery service
End-user
Data
Data
End-user applications

Figure 3. The prototype system architecture. WSO2 Enterprise Service Bus instances run on Linux nodes on the Amazon EC2 compute cloud, quality control library and Esper Complex Event Processing (CEP) software are distributed to Amazon EC2 nodes using the Storm distributed computing platform, while data storing is taken care of by Windows Azure Tables and PostGIS database.

Various data sources were connected to the prototype to test the ability to *integrate various distributed systems*. The data sources included a weather station provided by Vaisala, various sensor networks measuring indoor air quality and building energy consumption, a pollen forecast model provided by the Finnish Meteorological Institute (FMI), a weather forecast model provided the FMI, and the water level forecast model provided by the Finnish Environment Institute (SYKE). The integration of various data sources tests the prototype's ability to handle various data types and data formats. The sensor networks provide time series data from fixed sensor locations, the pollen forecast delivers gridded data for multiple forecasted time instances, and the water level forecast provides predicted future values for hundreds of fixed point locations. Also, various data sources test the ability to handle various interface protocols, including standards, such as OGC, and proprietary interfaces. The data source proxies in the architecture transform the incoming messages into the bus internal message format and take care of the communication with the actual data source.

The *extensibility* was demonstrated for the indoor air quality and building energy consumption measurement data sources. A data source proxy was implemented to transform indoor measurements from one data source into the bus internal data format, which was then used to deliver the data to the end-user applications for visualization. As new indoor data sources of the same data type were integrated, only a new data source proxy was implemented. No other changes in other components or the end-user applications were needed for the application to utilize the new data source.

The *configurability* was realized by using the workflow scripts provided by the WSO2 ESB. For example, in one test case the data were directed to the data storage without quality control checking, and in another case the same data messages were directed first to the data quality control, then to anomaly detection and to data storage at the same time.

The complex event processing was implemented using an open source event stream processing and event correlation engine called Esper, developed by EsperTech. Esper provides a specific language for expressing event conditions, called Event Processing Language (EPL). The anomalies or events that the users are interested in are transformed into EPL sentences and subscribed to Esper. Quality control was implemented using a proprietary quality control software library that includes rudimentary statistical tests and algorithms for data quality purposes (e.g. average, standard deviation, minimum, maximum, histogram, linear regression, FFT). In the prototype, detected events about quality problems or interesting anomalies can be delivered to end-users using a notification service that uses e-mail, HTTP and SMS protocols for sending notification messages.

3.2. Data storage and security

The *data storage* was implemented using commercially available noSQL cloud storage service Microsoft Azure Tables and an open source PostGIS SQL database. Azure Tables is an example of a noSQL database which does not use structured query language for data manipulation. The database provides scalable solutions for storing large data volumes with 99.9% uptime SLA (Service Level Agreement). However, noSQL databases are not optimal for handling geo-referenced environmental data, and therefore the prototype architecture also relies on a PostGIS database to store the geographical parameters. For example, the coordinates of a sensor can be stored in a spatial database that enables efficient spatial queries, but the observation time series is stored in a noSQL database that can handle larger amounts of data efficiently. The data can be retrieved from the prototype via a query service interface that hides the data storage implementation. Some part of the data, such as coordinates, may come from the PostGIS database, and some part, such as the observation time series, may come from the noSQL cloud storage. The prototype data storage costs were not considered, as the prototype concentrates on near real-time data processing.

The *data security* was implemented using the WSO2 Identity Server (IS) and the WS-Security based authentication mechanism of the WSO2 ESB. The Identity Server is used for user accounting, authentication and authorization, while the WS-security enables sending user credentials inside web service procedure calls (SOAP header). HTTPS protocol is used to encrypt network communication between the client and the Identity Server. IS supports the eXtensible Access Control Markup Language (XACML) which defines a declarative access control policy language implemented in XML and a processing model describing how to evaluate authorization requests according to the rules defined in policies. These mechanisms provided the prototype with good enough data security to prevent unauthorized access to the data or events generated or mediated by the system. However, thorough tests for attacking the system were not conducted during this study.

3.3. Performance and scalability

The performance and *scalability* of the prototype architecture was tested by measuring the message throughput as a function of computing nodes (Aalto, 2012). The test data sources connected to the prototype did not provide a data volume close to the system limits, so the system was overloaded with simple generated test observation messages. It appeared that an ESB implementation using WSO2 with ActiveMQ message queue in one node could handle around 1,800 messages (test messages of 470 bytes) per second, i.e. transforming the messages into bus internal format and redirecting them forward. The tests were run using Amazon on-demand M1 large instances which each have 7.5 GB of memory, and processing power of 4 EC2 units (one EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor) with costs of $0.26/hour for the EU region (Amazon Web Services, Inc., 2013). The performance of message digestion can be improved by adding more ESB nodes, and Amazon EC2 even provides a service for load balancing, i.e. new nodes can be engaged or stopped automatically whenever incoming traffic increases or decreases. In addition to adding new ESB nodes, the prototype also enabled scalability by utilizing the open source distributed real-time computation system Storm to distribute the computation of quality control and complex event processing.

We found out that in simple message processing cases the main bottleneck of the system is the ESB implementation. A scalability test was run with four nodes in the Storm cluster for complex event processing, and with one to four nodes for the ESB message mediation. The results show that the system scales up quite linearly as a function of the number of ESB nodes, at least with the tested number of nodes. The system could handle on average around 1,800 messages per second using one node, 3,400 using two, 5,300 using three and 7,000 using four ESB nodes. Thus, with four plus four nodes set up, the prototype can do simple event processing for around 7,000 observation messages per second with processing costs of around $2/hour.

In addition to processing costs, also the data transfer costs should be taken into account in near real-time processing. For example, Amazon has various options depending on from where the data come from and to where the data go. As the purpose of the architecture is to deliver only the minimum amount of data to end-users, we can firstly consider the costs of incoming dataflow. The alternatives for incoming data transfer costs are $0/GB and $0.01/GB (as of Oct 2013), so inputting 7000 messages (470 bytes each) per second costs $0-$0.12/hour. Transferring all the messages after event processing would cost $0/GB, $0.01/GB, $0.02/GB or $0.12/GB (as of Oct 2013) depending on the delivery location, so the output costs would be $0/hour, $0.12/hour, $0.24/hour, or $1.42/hour.

4 Discussion

Despite of recent developments in the information technology and standardization of data interfaces the efficient development of near real-time environmental monitoring applications utilizing distributed data sources needs careful engineering and could be supported by improved tools, design patterns, best practices, and architectural models.

Different architectures have been presented earlier for near real-time environmental data processing utilizing Enterprise Service Bus, complex event processing and cloud services. For example, Cao (2009) proposed use of the Enterprise Service Bus (ESB) architectural model to handle the data collection from distributed sensors developed by a variety of manufacturers relying on various standards; Motwani et al. (2010) discussed use of the ESB architectural model and complex event processing (CEP), so called event-driven SOA, as a means of achieving extensibility and interoperability in a heterogeneous environmental sensor network; and Suakanto et al. (2012)

proposed an architecture using cloud computing in sensor data processing for disaster early warning.

Unlike previous architectures, the proposed architecture combines the three technologies to fulfill the general requirements for distributed near real-time environmental monitoring presented in chapter 2.1. The prototype implementation showed that the proposed architecture can be used to implement environmental monitoring applications that take into account the general requirements. The prototype could *integrate various distributed systems* such as data sources and processing components, and be *extensible*, *configurable*, *scalable*, and *data secure*. The prototype was connected to various data source interfaces and data formats, including OGC-compatible data sources, but no OGC-compatible way to retrieve data was yet implemented. However, the architecture is not restricted to any specific data or interface formats, so fully OGC-compatible data processing chains are possible.

The requirement of *delivering only relevant information* was made possible by using anomaly detection, complex event processing and a notification service. The linear scalability suggests that the processing power intensive requirements for *timely delivery* and *quality control* can be achieved by adjusting the amount of nodes sufficient for the application-specific requirements. The *operational reliability* was relying on the cloud services for processing and data storage. Although no thorough testing of the reliability had yet been done the cloud services were up and running constantly during the prototype development and testing. More experiences are being gathered as the prototype continues running and further tests are implemented.

The *cost-efficiency* of the prototype relies on the usage of open source components and the cost-efficiency of the cloud computing services. However, by reducing the complexity of the computation, the same message throughput could be achieved with less processing power, i.e. with lower costs. The performance of a distributed data processing chain is dependent on the performance of individual components and services constituting the chain, but also on architectural design choices and the performance of selected software components. XML is a natural choice for use in interfacing external services such as the data sources and end-user applications, and XML was also chosen for the bus internal communication between the services. However, the marshaling of XML messages to java classes and vice versa requires additional processing time, which causes latency in the overall data processing. The performance could be improved by using more concise messaging formats and more efficient message marshaling. However, XML gives advantages that were seen as more important than the efficiency in implementation. During the development and the operation of the system the extensibility and human-readability of XML enables easier debugging, evolution and maintenance of the messaging. If the speed of processing should be maximized in a specific case, this can be done by using simpler messaging formats.

The performance testing was done with simple messages and simple event handling for the messages; in this case the ESB appeared to be the bottleneck of message throughput. However, as the complexity of messages or the complexity of processing of messages increases, the processing time for the message routing reduces compared to the processing of the data that the messages carry. The size of messages and the amount of processing depend on the data sources and the applications of the data. For example, sensor data sources usually send small messages frequently, as forecast models produce large amounts of data more seldom. The prototype implementation uses distribution of the computation separately for the ESB message mediation and for the data processing required by the quality control and the anomaly detection. Thus, the processing power can be adjusted according to the message frequency and amount of data processing of the data.

In some cases, because of high confidentiality of the data, the use of commercial cloud services for data processing and storing might not be acceptable. For example, data related to national security or emergency management is kept on secured servers instead of commercial cloud platforms. However, we see that a majority of environmental monitoring applications could rely on commercial cloud services and get a better, more secure, and more cost-efficient service than could be achieved by proprietary solutions.

At this stage, the study concentrated on the near real-time data processing performance. The processing and understanding of current observations may also require looking back at the historical values, thus emphasizing the requirement for storing and accessing data efficiently. In the next phase, the performance of the data storage and retrieval are tested more thoroughly.

5 Conclusions

Timely environmental information is relevant for decision-making in dynamic environmental conditions, and new environmental monitoring data sources are opening for applications. This paper presents general requirements for a distributed near real-time environmental monitoring system and suggests an architectural model for developing a system to support near real-time environmental monitoring applications using distributed data sources. The architecture utilizes state-of-the-art information technology architectural models and components, such as the Enterprise Service Bus (ESB) to control the data processing chain workflow and data security, complex event processing (CEP) to minimize data delivery to end-users, and cloud computing and cloud storage systems for scalable, robust and cost-efficient system implementation. The prototype system verifies the architecture and shows that the combination of ESB, complex event processing (CEP), cloud computing and distributed processing can solve the requirements for near real-time environmental monitoring, and be used to develop robust and powerful environmental data processing chains cost-efficiently.

6 Abbreviations

CEP	Complex event processing
EC2	Amazon Elastic Compute Cloud
ESB	Enterprise Service Bus
FMI	The Finnish Meteorological Institute
GEO	Group on Earth Observations
GEOSS	Global Earth Observation System of Systems
GMES	Global Monitoring for Environment and Security
HTTPS	Hypertext Transfer Protocol Secure
IS	Identity Server
OGC	Open GIS Consortium
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SWE	Sensor Web Enablement
SYKE	The Finnish Environment Institute
WS-security	Web Services Security
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language

7 References

Aalto A., 2012. Scalability of Complex Event Processing as a part of a distributed Enterprise Service Bus. Master’s thesis, Degree Program of Computer Science and Engineering, Aalto University, Espoo.

Amazon Web Services, Inc., 2013. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/ (accessed Oct 2013).

Buyya, R., Chee Shin Yeo, Venugopal, S., Broberg, J., Brandic, I., 2009. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 25 (6): 599-616.

Cao, J., 2009. Flexible data collection over distributed sensors on Enterprise Service Bus. In: 2009 International Conference on Frontier of Computing Science and Technology, Shanghai, 17-19 Dec 2009.

Chappell, D., 2004. Enterprise Service Bus. O'Reilly Media Inc.

Conrad, C., Hilchey, K., 2011. A review of citizen science and community-based environmental monitoring: issues and opportunities. Environmental Monitoring and Assessment 176: 273-291.

Erl, T., 2005. Service-oriented architecture: concepts, technology, and design. Prentice Hall Professional Technical Reference.

European Parliament, Council, 2007. Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community (INSPIRE), Brussels, Belgium.

GEO secretariat, 2010. Report on progress. GEO Secretariat, Geneva, Switzerland.

Hill, D.J., Minsker, B.S., 2010. Anomaly detection in streaming environmental sensor data: A data-driven modeling approach. Environmental Modelling & Software, 25(9): 1014-1022.

Kotovirta, V., Karvonen, J., von Bock und Polach, R.; Berglund, R., Kujala, P., 2011. Ships as a sensor network to observe ice field properties. Cold Regions Science and Technology 65(3):359 – 371.

Lee, K., Murray, D., Hughes, D., Joosen, W., 2010. Extending sensor networks into the Cloud using Amazon Web Services. In: IEEE International Conference on Networked Embedded Systems for Enterprise Applications (NESEA), Suzhou, 25-26 Nov. 2010.

Li, H., Wu, B., 2011. A Service-Oriented Architecture for Proactive Geospatial Information Services. Future Internet 3(4): 298-318.

Lim, S.B., Yoon, K., Eo, Y.D., 2012. Ubiquitous air quality monitoring system with service oriented architecture middleware. Journal of Convergence Information Technology 7(6): 193-201.

Luckham, D., 2002. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Professional.

Motwani, R., Motwani, M., Harris, F., Dascalu, S., 2010. Towards a scalable and interoperable global environmental sensor network using service oriented architecture. In: Proceedings of the 2010 Sixth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2010), Brisbane, 7-10 Dec 2010.

Plale, B., Gannon, D., Brotzge, J., Droegemeier, K., Kurose, J., McLaughlin, D., Wilhelmson, R., Graves, S., Ramamurthy, M., Clark, R.D., Yalda, S., Reed, D.A., Joseph, E., Chandrasekar, V., 2006. CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting. Computer 39(11):56-64.

Quinn, N.W.T., Ortega, R., Rahilly, P.J.A., Royer, C.W., 2010. Use of environmental sensors and sensor networks to develop water and salinity budgets for seasonal wetland real-time water quality management. Environmental Modelling & Software 25(9):1045–1058.

Rönkkö, M., Kotovirta, V., Karatzas, K., Bastin, L., Stocker, M., Kolehmainen, M., 2012. Proactive Environmental Systems: the Next Generation of Environmental Monitoring. In: Seppelt, R., Voinov, A.A., Lange, S., Bankamp, D. (Eds.): iEMSs 2012 International Congress on Environmental Modelling and Software, Leipzig, Germany, 1-5 July 2012.

Suakanto, S., Supangkat, S., Suhardi, Saragih, R., Nugroho, T., Nugraha, I.G.B.B., 2012. Environmental and disaster sensing using cloud computing infrastructure. In: Proceedings of the 2012 International Conference on Cloud Computing and Social Networking (ICCCSN), Bandung, West Java, Indonesia, 26-27 April 2012.

Usländer, T., Jacques, P., Simonis, I., Watson, K., 2010. Designing environmental software applications based upon an open sensor service architecture. Environmental Modelling & Software 25(9):977-987.

Williams, M., Cornford, D., Bastin, L., Jones, R., Parker, S., 2011. Automatic processing, quality assurance and serving of real-time weather data. Computers & Geosciences, vol. 37(3): 351-362.

Wächter, J., Babeyko, A., Fleischer, J., Häner, R., Hammitzsch, M., Kloth, A., Lendholt, M., 2012. Development of tsunami early warning systems and future challenges. Natural Hazards and Earth System Sciences, 12:1923–1935.

Yao, Y., Sharma, A., Golubchik, L., Govindan, R., 2010. Online anomaly detection for sensor systems: A simple and efficient approach. Performance Evaluation 67(11):1059-1075.