Flexible and Efficient Sensor Data Processing – A Hybrid Approach

Claas Busemann, Christian Kuka, Daniela Nicklas, Susanne Boll

OFFIS - Institute for Information Technology
Oldenburg, Germany
busemann|kuka@offis.de
http://www.offis.de

Carl von Ossietzky Universität
Oldenburg, Germany
daniela.nicklas|susanne.boll@uni-oldenburg.de

Abstract: The integration of various sensor data into existing software systems is becoming increasingly important for companies and even private users. As the number of embedded devices of all sorts (sensors, mobile phones, cameras etc.) also constantly increases, the development of flexible sensor applications gets more and more difficult. These applications have to handle a large number of sensors transmitting their data in various formats using different protocols. Middleware technologies are a good way to hide the complexity of communication protocols and data processing from the application. However, integrating efficient sensor data processing into a middleware requires several design choices which depend on the planned applications. Usually such systems are either efficient, allowing the processing of large numbers of data streams, or flexible, allowing the easy modification of the processing during runtime. In this paper a hybrid approach is introduced combining the benefits of two popular processing technologies, Service Oriented Architectures (SOA) and Data Stream Management Systems (DSMS), and by that enable the processing of large numbers of data streams while at the same time the system can be flexibly modified. Therefore these two technologies are analyzed to identify the benefits and disadvantages depending on their flexibility and efficiency.

1 Introduction

Today, many low cost and easy to install sensors are available, many more in the future. However, to realize even simple sensor-based applications, the effort is quite high: there are many different interfaces and standards to communicate with the sensor systems. The data read from the sensors is of variable quality and has often to be interpreted, aggregated or otherwise preprocessed before being usable by applications. Since the heterogeneity of sensor systems is so high, applications are bound to certain vendors or proprietary so-called standards. Hence, even if more and more sensor systems are already installed, it is hard to re-use the effort of pre-processing for other applications. Thus, future middleware

architectures for sensor-based applications have to fulfill two major requirements that are often contradicting: flexibility and efficiency. One solution for the flexible orchestration of reusable sensor processing services could be a service-oriented architecture using the Sensor Web Enablement [Ope] framework. Web service technology has many benefits regarding extensibility and standardization. They mostly rely on an event driven processing paradigm, where active services push business events on a service bus and other services subscribe to these basic events to provide higher level information (like complex event processing engines). Finally, the end-user application receives only those events it is really interested in.

When dealing with raw sensor data, this paradigm gets to its limits. Not every sensor measurement is a meaningful event, and the throughput of event buses is not suited for high loads of input data, as we can show in our evaluation. Hence, to close the gap between the high level service oriented world and the low level sensor network world we introduce an hybrid approach: for pre-processing high loads of sensor data, a data stream management system is used which acts as a configurable service within a service oriented architecture. Our evaluation shows that this approach dramatically increases the possible amount of incoming sensor data. The remainder of this paper is organized as follows: Section 2 discusses the related work. In Section 3 two popular sensor data processing technologies are analyzed followed by the description of our hybrid approach in Section 4. Section 5 contains a performance evaluation of the approach. Finally, section 6 draws a resume.

2 Related Work

Sensor data processing has been integrated into several middlewares like the Oracle Sensor Edge Server [Ora10b], the SensWeb platform [Mic08, SNL+06], the WISE platform [PHHL04] or the FireEagle platform [Yah07]. These systems use service oriented architectures (SOA) to realize the sensor data processing. This allows the developers to process the incoming sensor data using services and after that provide it to the application. SOA based processing systems are very flexible as they allow the simple integration of application specific services and can easily be orchestrated. User access administration can also be easily integrated, as every service is able to check the access permissions. However, a SOA based system usually cannot handle high-volume streams of incoming sensor data because of the administration overhead that goes along with service oriented architectures.

Other sensor middlewares like the Nexus platform [GBH+05], SStreaMWare [GRL+08] or the GSN middleware [SA07] are based on Data Stream Management Systems (DSMS). A DSMS is able to process high-volume streams of data in a fast and efficient way. Therefore those systems provide the same extended relational algebra operators like database management systems (DBMS). This algebra is extended through a temporal algebra to perform aggregation and joins on datastreams. In the OpenSource Community there exist a wide range of DSMS implementations like Esper [Esp08] from Espertech, IEP [iep] from the OpenESB community or TelegraphCQ [CCC+03]. As there is no public standard for DSMS query languages, all these implementations are using their own query language which are usually based on SQL. However, integrating application specific operators into

DSMS can be extremely complicated as every new operator has not only to be integrated into the DSMS but also into the query language. Adding user access administration to a DSMS can also be very complicateted, as most DSMS are not designed to handle user access permissions.

3 Sensor Data Processing

In this section, two popular data processing technologies, Service Oriented Architectures (SOA) and Data Stream Management Systems (DSMS), are analyzed. This is done by evaluating existing open source systems which are based on one of these technologies. As DSMS and SOA based processing technologies are significantly different from each other the evaluation is separated for each of these. Finally, the features of both technologies are compared to each other.

3.1 Service Oriented Architectures (SOA)

SOA based data processing systems are analyzed by identifying the benefits and disadvantages of three popular open source SOA based systems. These are Service Mix 4 [Apa10] developed by Apache Group, OpenESB [Ora10a] developed by Sun and PEtALS [OW210] developed by OW2. As all of these systems are based on a service oriented architecture, they all allow the simple definition of data processing. Therefore, the developer chooses the services she needs to process the incoming sensor data. These can be services executed inside of the SOA but also external services that run on a different server. After that she defines in which order these services are used by linking them logically together. However, there are several features making SOA based systems extremely flexible. As they usually follow the ”publish, find, bind” principle using a service broker they are able to discover additional services at runtime. SOA systems usually can be clustered. This makes it possible to run services with high memory or CPU usage on more powerful machines. The deployment of new services without interacting with the running systems (hot deployment) is not supported by some systems. Some systems can be integrated into application servers . This can be very useful when building web based sensor applications. Other systems are able to run as a standalone application which makes it easy to integrate them into ordinary applications. The orchestration of services is often done using BPEL [WCL+05] which is a widely accepted standard. Some systems are able to handle access permissions to single services.

However, not all of these features are supported by every SOA implementation. Tab. 1 shows an overview of analyzed systems and the features they support.

The service oriented approach brings a lot of flexibility but it also slows down the processing speed because of the administrative overhead. This is caused by the service broker, which has to decide what service to use next. The use of external services also slows down the system because of the communication overhead. If services are using different

	ServiceMix 4	OpenESB	PEtALS
Clustering	+	+	+
Service Broker	+	+	+
Hot Deployment	+	-	+
Standalone	+	-	+
Embedded	-	+	-
Access Perm.	+	-	+
Orchestration	Camel, EIP	BPEL	BPEL

Table 1: SOA based Systems - Feature Overview

communication protocols, the conversion also increases the processing time. Beyond that, external services can not be controlled as the developer has no influence on their behavior.

3.2 Data Stream Management Systems (DSMS)

Data Stream Management Systems (DSMS) are designed to process high-volume streams of data in a fast and efficient way. Therefore those systems provide the same extended relational algebra operators like database management systems (DBMS) extended through a temporal algebra to perform aggregation and joins on data streams. However, DSMS still differ from each other by features and algebra. In this section three popular open source DSMS are analyzed: TelegraphCQ [CCC+03] developed by the Berkeley University, Odysseus [JBG+09] developed by the University Oldenburg and Esper [Esp08] developed by EsperTech. As mentioned before, DSMS are usually controlled using a relational algebra. This algebra is translated into a logical operator graph which is deployed by the system. This graphs can be optimized by the system to make the processing more efficient. Some DSMS support the static optimization of operator graphs, which is done before the graph is deployed. Other systems are able to dynamically optimize the graph while the system is running. Clustering is also supported by some of the systems, but the implementations differ from each other. Odysseus for example realizes this by sharing operators on multiple Odysseus instances using a P2P network, while TelegraphCQ technically builds up a shared database which makes it possible to process the operator graph on different instances of the system. DSMS can use different scheduling strategies to optimize the memory and CPU usage. Some systems are able to use graph scheduling, which means that every operator graph runs as an own thread and operators are processed sequentially. Other systems allow operator scheduling, where every operator runs in its own thread. There are also systems that allow hybrid scheduling in which case the single operators or groups of operators run in an own thread. Another important feature is the prioritization of operations inside the system. This can be handled by allowing to prioritize a request to the system or allowing the prioritized processing of selected data by adding a priority to the data tuples. Complex Event Processing (CEP) is used to analyze data streams and search for patterns. As there is no public standard for DSMS, query lan-

guages like SQL for DBMS all the analyzed systems use their own query language which are based on SQL. TelegraphCQ uses the Continues Query Language (CQL), Esper the Event Pattern Language (EPL) and Odysseus SPARQL (for RDF tuple streams) and sSQL (for relational tuple streams). Tab. 1 shows an overview of analyzed systems and the features they support.

	TelegraphCQ	Odysseus	Esper
Static Opti.	-	+	-
Dynamic Opti.	+	-	-
Clustering	+	+	-
G. Scheduling	+	+	+
O. Scheduling	+	+	-
H. Scheduling	-	+	-
Prioritization	-	+	+
CEP	-	+	+
Query Language	CQL	sSQL	EPL

Table 2: DSMS Systems - Feature Overview

The data stream management approach brings a lot of efficiency when filtering and aggregating data but also lacks the possibility to extend the available operators. This is caused by the underlying system which is designed to process data while minimizing memory and CPU usage. As the system has to know each operator at start time to be able to optimize the operator graph and choose the right optimization strategy, the integration of new operators is very complicated.

3.3 Service Oriented Architectures vs. Data Stream Management Systems

The previous sections show that depending on their features SOA based systems and DSMS are primarily different. However, they still share the same application purpose which is processing incoming data using predefined operators. Because of the underlying architecture, SOA systems are extremely flexible. The main benefits of this architecture are the flexible orchestration of services using accepted standards like BPEL, that services can easily be added and modified, that the integration of external services is very easy and that access permissions to services can be handled. The main disadvantage is that the SOA slows down the processing time. Also an optimization of the operator graph is not possible in a way comparable with the optimization inside a DSMS. External operators are easy to integrate but they are also hard to control. This can effect the data processing as they may stop working or change their behavior.

DSMS on the other side are extremely efficient. The main benefits of this architecture are that it is possible to process data in a memory and CPU usage efficient way, that operator graphs can be optimized, that the processing can be optimized using different scheduling strategies, that queries can be prioritized and that complex events can be recognized. The

main disadvantages are that the integration of new operators is difficult, that operators can not be deployed during runtime, that the integration of external operators usually is not possible, that no standard query language exists and that access permissions to operators can not be handled.

However, sensor data processing for applications which have to be flexible and at the same time handle a huge amount of sensor data in a memory and CPU usage efficient way should still be possible. Thats why we introduce our hybrid approach, which is explained in detail in the next section.

4 Hybrid Approach

In this section, a hybrid system is introduced which is based on a SOA system and uses a DSMS for efficient processing of sensor data. The basic idea is to reduce the number of data the SOA system has to handle by processing standard operations inside of a DSMS. A pure DSMS can not be used as it can not provide the flexibility that comes along with the SOA system. A schematic view of the system can be found in Fig. 1. To realize this hybrid system, the DSMS is integrated into the SOA system. Therefore a service container is built around the DSMS. The service container is bidirectional connected with a bus system that connects all services in the SOA. This allows the integration of the DSMS into the SOA system as one of its services. To enable the communication between the SOA services and the DSMS operators the sinks and sources of the DSMS are mapped as provider and consumer endpoints for other services of the SOA. A provider endpoint receives sensor data from other services. Each incoming message is transformed into the internal format and pushed into the physical processing plan of the DSMS. Thereby the timestamp of the messages are preserved for usage in timewindow operators. A consumer endpoint publish the processed sensor data on the bus system for other services. Therefore a new message that includes the processed measurements is generated every time the DSMS produce a new output. This means that other services can not request older values. If they are subscribed to the consumer endpoint they get the newest values from the bus system instead. The service container creates those endpoints when a new sink or source is registered through a processing query and publishes these endpoints for other services. The container is also responsible for the conversation of messages from a provider endpoint to a DSMS source and from a DSMS sink to a consumer endpoint. Sensor data can still be sent directly to the DSMS. Therefore the system allows the communication with the DSMS using a TCP/IP connection. This makes it possible to pre-process data even before the SOA has to handle any of it.

As it now has to be possible to deploy a shared operator graph, which includes operators of the SOA and the DSMS, a framework is built around the system which allows the description, storage and deployment of shared operator graphs.

The description of the shared operator graphs is realized using XML. Operators are always based on operator types which are predefined by the system. One of these operator types is the “DSMS operator”. Every other operator type represents an operator of the SOA. When

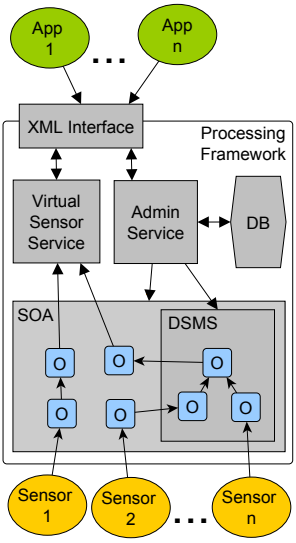


Figure 1: Schematic View of the Hybrid System

creating an operator based on an operator type using the XML interface the operator is first saved in a database. The system also allows the creation of inputs and outputs. Inputs have to define the ID of a sensor data stream and are used to transmit data into the processing framework. Outputs have to define a unique name and are used to receive processed data out of the processing framework. The user then has to orchestrate the inputs, outputs and operators by linking them together. After all operators and links are saved in the database, the user can use the XML interface to deploy the operator graph. The "Admin Service" of the processing framework then accesses the database and creates the orchestration for the SOA system and the DSMS. If the user decides to change the operator graph she can always change its database representation. After that the graph has to be redeployed. After the operator graph is deployed, the incoming sensor data is received by the SOA or the DSMS. The incoming data then is processed by the SOA and DSMS operators depending on the deployed orchestration. After the sensor data is processed it can be accessed by the application using the "Virtual Sensor Service".

5 Performance Evaluation

In this section a SOA based system, a DSMS and the hybrid approach are compared using a performance evaluation. As the evaluation implementation of the hybrid approach is based on Esper and ServiceMix 4 these systems are also used as stand alone systems for the evaluation.

5.1 Evaluation scenario

The evaluation scenario is based on a community web application that allows the users to monitor the position of their pet.

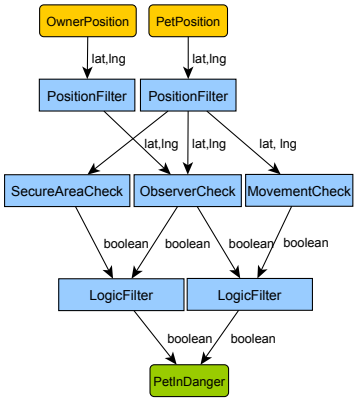
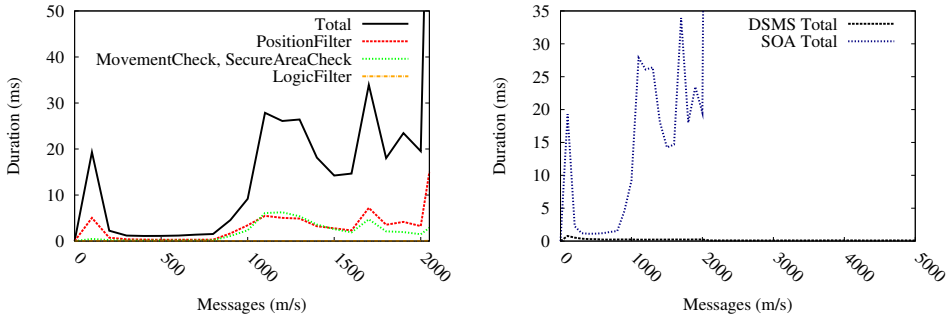


Figure 2: Schematic View of Petfinder Processing

Fig. 2 shows a schematic view of the processing for the petfinder application. Position information is transmitted from the pet and the owner. The position filter (PositionFilter) is used to filter invalid data. After that the valid position data is transmitted to operators which check if the pet is in danger. They check if the pet is in a secure area (SecureAreaCheck), if it is observed by its owner (ObserverCheck) and if it has moved over the last time (MovementCheck). After that, these operators transmit a boolean value that indicates if the pet may be in danger or not to logic filters (LogicFilter). The logic filters make sure that no alarm is sent while the pet is observed. They also check if the events occurred at the same time. If the "PetInDanger" operator receives "true" from one of these filters the pet is in danger and a alarm message has to be sent.

The evaluation is done using a worst case scenario. Pets are never observed, never in a secure area and do not move. This forces the processing system to handle a huge amount of sensor data as every incoming sensor value has to be processed by every operator. It starts with 100 owners monitoring 100 pets. Owners and pets send a new position every five seconds. All data is received during one second giving the system 4 seconds to process the data before new data is transmitted. The number of owners and pets is increased by 100 every 50 seconds. 50% of the incoming pet and owner positions are illegal and have to be filtered. The performance of each system is detected by measuring the average time between transmitting and receiving a single data value. The test system consist of a Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93GHz running Linux Debian with 2.6.32-3-amd64 Kernel and 4GB RAM. The Java(TM) SE Runtime Environment version 1.6.0_20 is used.



(a) Performance Evaluation of ServiceMix 4

(b) Performance Evaluation of Esper

Figure 3: Performance Evaluation of Single Systems

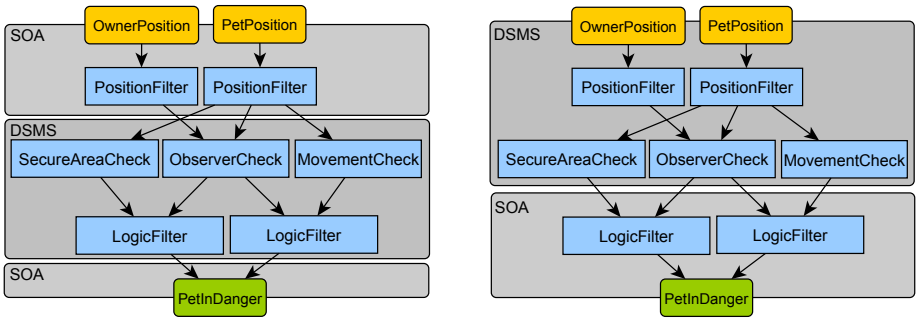
5.2 ServiceMix 4

ServiceMix 4 is tested using the release version (4.0.0). As ServiceMix 4 does not have any operators needed for the application scenario the operators are self implemented. Because ServiceMix provides the developer with a simple interface structure when programming operators and the integration is as simple as putting a jar file into a folder, the necessary operators can be programmed and integrated within one day.

The results of the performance evaluation can be found in Fig. 3a. The peak at the beginning is a result of the binding procedure of the services when the first message is processed. But it shows that ServiceMix 4 is easily able to handle about 700 incoming data streams within 1 seconds processing time. At that point every sensor value is processed in about 2ms. As soon as the number of data streams comes near to 1.000, the systems is no longer able to handle them without extremely slowing down. The system now needs about 30ms to handle one data stream. When more than 2.000 data streams are processed, ServiceMix 4 is no longer able to handle the amount of data and stops processing. This is caused by the bus of ServiceMix which is only able to handle a fixed number of data streams. Fig. 3a also shows the processing time of each operator. However, the evaluation shows that most of the processing time is needed for the communication between the operators, as the sum of the processing time of all operator is often about 10 ms below the total processing time.

5.3 Esper

Esper is evaluated using the release version 2.2.0. All needed operators are already integrated into Esper. The orchestration is done by writing a EPL request. The results of the performance evaluation can be found in Fig. 3b. It shows that Esper can easily handle up to 5.000 in 1 seconds processing time. Every sensor value is processed in about 1ms what is far below the time that ServiceMix 4 needed. Because of the closed system approach of



(a) Evaluation Scenario 1

(b) Evaluation Scenario 2

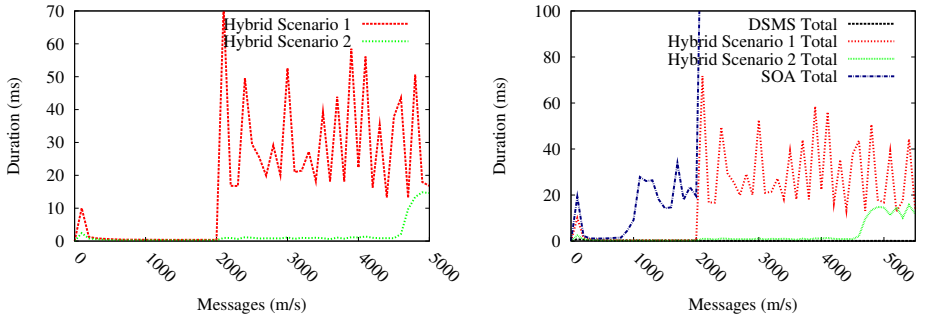
Figure 4: Evaluation Scenarios

Esper it is not possible to monitor the processing time of a single operator without spoiling the evaluation results. However, as the implementation of the ServiceMix operators does not differ too much from the implementation of the Esper operators it should be clear that the time benefit is achieved by the missing communication overhead.

5.4 Hybrid Approach

The hybrid approach is tested using the basic architecture described in section 4. Therefore the operator graph is split into parts. These parts are either processed by the ServiceMix 4 or Esper. The communication between these parts is handled by the ServiceMix 4. Two different shared operator graphs are evaluated. The first one can be found in Fig. 4a.

In this evaluation scenario, the "SecureAreaCheck" operator, "ObserverCheck" operator, "MovementCheck" operator and the two "LogicFilter" operators are handled by the DSMS. The processing of these operators should be much faster than the rest, because of the missing administrative overhead. The PositionFilter operators are still part of the SOA system and can therefore easily be changed or replaced during runtime. The second evaluation scenario can be found in Fig. 4b. The SOA is only used for the "LogicFilter" operators. Everything else is handled by the DSMS. In this scenario the SOA only has to handle a minimum of data streams as the DSMS already reduces their number using the "PositionFilter" operators. The results of both evaluations can be found in Fig. 5a. The result of the first evaluation scenario shows, that the processing of a part of the operator graph inside a DSMS reduces the average processing time of every data stream. However, the processing still gets pretty slow when more than 2.000 data streams are processed. This is exactly the point where ServiceMix 4 stopped working in the prior evaluation. The second evaluation scenario shows that even more data streams can be processed if the DSMS is used to filter data streams, which do not have to be processed by the SOA. As the "PositionFilter" operators which reduce the amount of data streams by 50% are now



(a) Performance Evaluations of Hybrid System

(b) Performance Evaluation of All Systems

Figure 5: Performance Evaluations

processed by Esper, ServiceMix only has to process a minimum of data streams. This reduces the communication overhead and allows the complete system to process about 4.500 data streams before slowing down. The average processing time is also extremely low at about 2 ms. The results of all tests can be found in Fig. 5b. The evaluation proves that using a hybrid system, which combines the benefits of a SOA system and a DSMS, is a good way to build a flexible and efficient data processing system. The hybrid system caused a drastically performance rise in both evaluation scenarios in comparison to SOA system. However, a pure DSMS seems to be the fastest system, but using the hybrid system operators can still be implemented inside of the SOA allowing them to benefit from the flexibility that comes along with those systems. At the same time selected operators can be integrated into a DSMS to reduce the communication overhead and increases the efficiency of the processing.

6 Conclusion

In this paper two popular data processing technologies, Service Oriented Architectures (SOA) and Data Stream Management Systems (DSMS), are analyzed. Their benefits and disadvantages have been identified and analyzed. Based on this evaluation a hybrid system has been developed that increases the performance of a SOA system by integrating a DSMS into it. As the DSMS dos not have to be used for all operations the hybrid system still provides all the flexibility that comes along with the SOA. The positive effects of this hybrid approach are proven by a performance evaluation.

References

[Apa10] Apache Software Foundation. ServiceMix 4. `http://servicemix.apache.org/`

home.html, 2010.

[CCC+03] Sirish Chandrasekaran, Sirish Ch, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertan World, 2003.

[Esp08] Espertech Inc. EsperTech: Event Stream Intelligence. http://camel.apache.org/manual/, 2008.

[GBH+05] M. Grossmann, M. Bauer, N. Honle, U. Kappeler, D. Nicklas, and T. Schwarz. Efficiently Managing Context Information for Large-Scale Scenarios. In *Proceedings of Pervasive Computing and Communications, IEEE Computer Society*, 2005.

[GRL+08] Levent Gurgen, Claudia Roncancio, Cyril Labbé, André Bottaro, and Vincent Olive. SStreaMWare: a service oriented middleware for heterogeneous sensor data management. In *ICPS '08: Proceedings of the 5th international conference on Pervasive services*, pages 121–130, New York, NY, USA, 2008. ACM.

[iep] IEP - Open Source Complex Event Processing (CEP) and Event Stream Processing (ESP) Engine . https://open-esb.dev.java.net/IEPSE.html.

[JBG+09] Jonas Jacobi, André Bolles, Marco Grawunder, Daniela Nicklas, and H.-Jürgen Appelrath. A physical operator algebra for prioritized elements in data streams. *Computer Science - Research and Development*, 2009.

[Mic08] Microsoft Research. SenseWeb Project. Technical report, Microsoft Research, Redmond, USA, 2008.

[Ope] Open Geospatial Consortium (OGC). Sensor Web Enablement (SWE). Specification.

[Ora10a] Oracle Corporation. OpenESB. https://open-esb.dev.java.net, 2010.

[Ora10b] Oracle Corporation. Oracle Sensor Edge Server. http://www.oracle.com/technology/products/sensor_edge_server/index.html, 2010.

[OW210] OW2 Consortium. PEtALS. http://petals.ow2.org, 2010.

[PHHL04] Rui Peng, Kien A. Hua, and Georgiana L. Hamza-Lup. A Web Services Environment for Internet-Scale Sensor Computing. In *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing*, pages 101–108, Washington, DC, USA, 2004. IEEE Computer Society.

[SA07] Ali Salehi and Karl Aberer. GSN, Quick and Simple Sensor Network Deployment. In *European conference on Wireless Sensor Networks*, 2007.

[SNL+06] Andre Santanche, Suman Nath, Jie Liu, Bodhi Priyantha, and Feng Zhao. SenseWeb: Browsing the Physical World in Real Time. In *Demo Abstract, ACM/IEEE IPSN 2006, Nashville, TN*, April 2006.

[WCL+05] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. 2005.

[Yah07] Yahoo. FireEagle: Centralized management of user location. Technical report, Yahoo Research, 2007.