Integrating and Processing Events from Heterogeneous Data Sources

Ivo Koga[1], **Claudia Bauzer Medeiros**[1]

[1]Institute of Computing – State University of Campinas (UNICAMP)
13083-852 – Campinas – SP – Brazil

{koga,cmbm}@ic.unicamp.br

Abstract. *Environmental monitoring studies present many challenges. A huge amount of data are provided in different formats from different sources (e.g. sensor networks and databases). This paper presents a framework we have developed to overcome some of these problems, based on combining aspects of Enterprise Service Bus (ESB) architectures and Event Processing mechanisms. First, we treat integration using ESB and then use event processing to transform, filter and detect event patterns, where all data arriving at a given point are treated uniformly as event streams. A case study concerning data streams of meteorological stations is provided to show the feasibility of this solution.*

1. Introduction

Sensor network applications present countless challenges in environmental monitoring. These networks can be composed by many sensors with different types, models and manufacturers. They can monitor many aspects of the environment, *e.g.*, motion, sound, temperature, humidity and light. An open problem is how to obtain, filter and validate data. As the amount of data sources and heterogeneity increases, the difficulty to manage environmental data also increases.

There are several approaches to deal with sensor data. At one end, proposals concentrate the processing at the sensor side, *e.g.*, defining sampling rates inside the sensor network (*i.e.* without using an external database). At the other end, studies concentrate on receiving all data and processing it at the server side. In-between, there is a vast range of studies, *e.g.*, creating hierarchies of processing/storage nodes.

On environmental studies, most of these solutions are concerned with solving heterogeneity issues – *e.g.*, when there is a set of distinct networks of different kinds of sensors providing data. Solutions vary from limiting the types of sensors (and thus having control over data sources) to defining some kind of filtering module that will smooth out differences in data values.

Our proposal is to attack the heterogeneity problem using Enterprise Service Bus (ESB) and Event Processing theory to deal with sensor data. Under this perspective, we consider sensor networks as event producers and manage their events with filtering, matching, derivation and consumption of events. We can then process not only the sensor data, but also events they raise, taking actions and selecting events to monitor.

Event processing for treating sensor data is normally found in industrial settings. Here, we adapt this to the environmental monitoring scenario. The main contribution is therefore to extend Event Processing theory to this context, and to combine it with an

architectural approach that supports integration of heterogeneous data streams (sensors and other sources).

2. Basic Concepts

2.1. Enterprise Service Bus

An Enterprise Service Bus (ESB) is a distributed infrastructure that uses messages and open standards to provide integration of systems [Menge 2007]. The ESB provides routing, invocation, mediation and other capabilities to facilitate integration. To integrate a new system using the ESB, it is only necessary to provide an adapter that fits the new system message format to couple with the ESB.

Since there are many different transport protocols used by systems, there are adapters to support the most common protocols (*e.g.* HTTP, FTP, SMTP) in most of the available ESBs. If a system provides messages using one of those transport protocols it is easier to integrate them. If it is not the case, one can implement a customized adapter which can be used and reused. According to [Rademakers and Dirksen 2009] some of the core functionalities of an ESB include location transparency, transport protocol conversion, message transformation (*e.g.* from SOAP to a custom XML format), message routing and enhancement, security, monitoring and management.

2.2. Complex Event Processing

Complex Event Processing (CEP) [Magid et al. 2010, Dunkel 2009] has several common features with Stream Processing [Babu and Widom 2001]. However, the latter usually focus on fast querying of data in streams while the former concentrates on extracting information from clouds of events (*i.e.*, lots of streams with many events not always ordered) [Luckham, D. C. 2006]. CEP deals with reaction rules such as the ones in active databases and also with events' causal relationships, providing ways to create multi-layered architectures of events.

In CEP, an event is an object signifying an activity with three aspects (Form, Significance and Relativity) that a computer can process [Luckham 2001]. Events can be matched by an *event pattern* that is a template that describes the event and all the appropriate context descriptions such as causal dependencies, timing, etc. Event Patterns can produce aggregation of events, creating a hierarchy with a sequence of levels. Each level has its own rules that specify how one can infer the higher layer events from lower level ones.

To process events, one has to deal at least with three entities: event producer, event consumer and event processing agent (EPA). The producer (*e.g.* a sensor network) introduces events into the system while the consumer (*e.g.* user application) receives events. In-between producer and consumer there is the EPA, which monitors system execution to detect patterns and process events.

[Etzion and Niblett 2011] proposed a hierarchy of EPA types. The higher layer is an abstract definition (Event Processing Agent). Below that abstract definition there is the Filter, Transformation and Pattern detect EPAs. A "Filter" EPA takes an event and decides whether it will be selected or not. It is used when there is need to remove irrelevant events from the event processing stream. When the event is incompatible with the filter

expression (*i.e.*, the expression cannot evaluate the event), it is considered non-filterable. To transform an event, *i.e.* take events and create different output events based on a derivation expression, [Etzion and Niblett 2011] show many possibilities. Consider the following sensor events written under the schema <Sensor ID, Timestamp, Temperature, Humidity, Air Pressure>:

- <1999, 10:00:00, 25, 57, 1025>
- <2000, 10:01:00, 25.2, 57, 1025>

Table 1 exemplifies the result of applying each EPA to these events.

Table 1. Transformation EPAs

	Transformation EPAs	Input	Output
1	Translate	Translate (<1999, 10:00:00, 25, 57, 1025>)	<1999, condition=normal>
	1.1 Enrich	Enrich (<1999, 10:00:00, 25, 57, 1025>, location)	<1999, location=Campinas>
	1.2 Project (temperature)	Project (<1999, 10:00:00, 25, 57, 1025>, temperature)	<1999, 25>
2	Aggregate	Aggregate (<1999, 10:00:00, 25, 57, 1025>, <2000, 10:01:00, 25.2, 57, 1025>, average_temp)	avg temp = 25.1
3	Split	Split (<1999, 10:00:00, 25, 57, 1025>, temperature, humidity)	<1999, 25>, <1999, 57>
4	Compose	Compose (<1999, 10:00:00, 25, 57, 1025>, <2000, 10:01:00, 25.2, 57, 1025>, humidity = humidity)	Humidity = 57

For instance, line 1 shows that the "translate" EPA eliminated attributes timestamp, temperature, humidity, air pressure and inserted a new attribute called condition, derived from the attributes in the input event. The "enrich" EPA can take input from the input event as well as from an external data source. The "project" EPA shows only some attributes from an input event, similar to the relational algebra project operation.

There is also the possibility to detect patterns. The Pattern Detect EPA type provides means to detect basic event patterns (logical operators, threshold, subset selection, etc) which do not depend on timing and ordering of events, and dimensional patterns (temporal, spatial and spatiotemporal) which rely on time, space or both.

3. Framework description

Our framework, shown in Figure 1, uses ESB and event processing to facilitate the integration of events produced by sensor networks and other types of data sources. At the left there are data providers. At the right, there are the data/event consumers that are interested in the output of our framework. Between them, steps (A, B, C, D, E, F and G) handle data integration and processing. All interactions between the framework producers and consumers can be held by ESB adapters (*e.g.*, *out1* and *outn* in Figure 1), steps "A" and "G".

Data providers and consumers provide and expect data in many different ways such as Web Services, database connections, file system, streams, etc, and using different protocols (*e.g.* SOAP, JDBC, FTP). To get/provide data, it is necessary to use each of these different types of communication possibilities. Since there are many available ESB adapters (for the most common types of protocols), the communication between the framework and data providers/consumers is easier.

Once data are obtained from the provider(s), it is possible to filter or continue processing them (step "B"). Filtering consists in deciding whether a message will be selected or not, *i.e.* if it will continue inside the processing flow or will be discarded – see Section 2. This is used when there is a need to remove irrelevant messages from the providers.

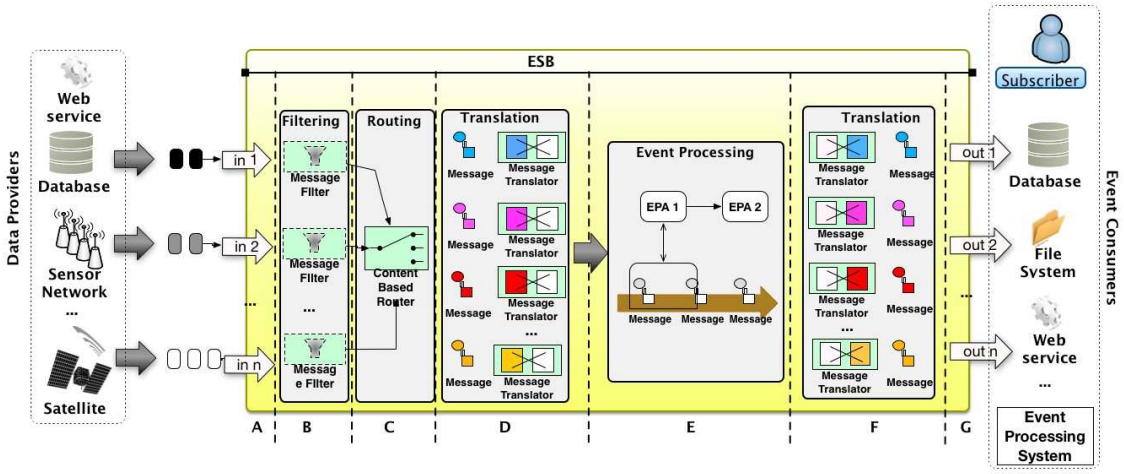

Figure 1. Architecture of the framework

After the filtering process, messages may have to be translated. To do that, the message passes through a *Content Based Router* (CBR) in step "C". A CBR is an Enterprise Integration Pattern [Hohpe and Woolf 2003] which reads the message content and routes the message to the appropriate message translator based on the message content.

Step "D" has some message translators to translate each different type of message format to a common message format inside the framework, following the suggestion of [Hohpe and Woolf 2003]. This homogenizes the data treated within the framework and provides another level of independency among other steps. Once messages are filtered and standardized, event processing, in step "E", can be applied (as discussed in section 2.2). Events can be processed by a network of EPAs and each EPA can nest several other EPAs.

Event consumers can be any human, software or machines interested in the events output by the framework. The output can be in many formats, so we put another translation step "F" so that the output can be translated into the consumer's expected format. Figure 1, at the right, shows examples of consumers: subscriber, database, file system, Web Service, another Event processing system, etc.

As an example, let us assume that we are retrieving data from a Web service and a sensor network using adapters *in1* and *in2* respectively (step "A"). Both providers are providing temperature sensor readings of the same place at the same time. Once the connection is established, only data that satisfy an SQL or an EPL (Event Processing Language) query go to the next step. After the filtering process (step "B"), in step "C" the CBR routes messages to the appropriate message translator (step "D"). Events can be detected by a pattern, *e.g.* a temperature property in the sensor stream different from a temperature property in the SQL query (step "E") indicates a possible failure. The framework translates the event message format to the subscriber expected format in step "F" and then sends a notification using an adapter, *e.g. out1* in step "G".

4. Case Study

Our case study concerns sensor data that cover over 30 types of variables, provided by the Cooxupé coffee cooperative, the largest coffee cooperative in the world. Figure 2 gives an

overview of data collection and transmission, with the Cooperative data collecting center at “C”. As shown in the Figure, this center centralizes data collection, and our lab, LIS, shown in “D”, retrieves sensor data from the center via its FTP interface, which cooxupé created for us.

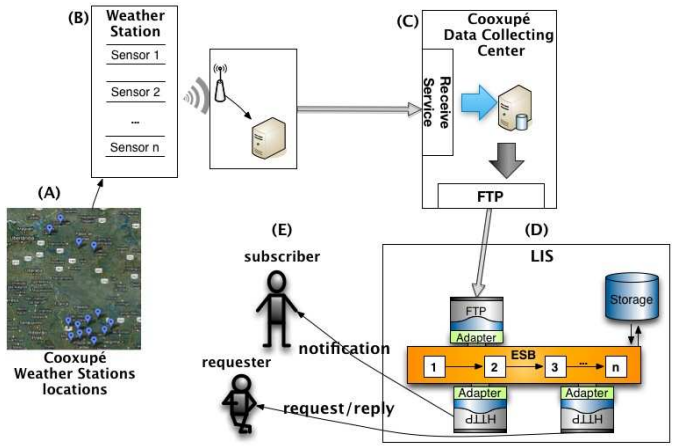

Figure 2. Case study: data processing workflow.

In more detail, sensors are deployed in weather stations, located at 13 locations in the states of Minas Gerais and São Paulo. Table 2 shows the 28 types of measurement collected by the Cooxupé weather stations and their description. Eleven of the 13 stations provide 28 types of measurement, and 2 stations (located at Guaxupé and Alfenas) provide 26 types of measurements (excluding internal moisture content and internal air density). They monitor meteorological variables at strategic locations, covering a large percentage of the areas where the cooperated farms are located[1].

The sensors at each station collect data at one hour intervals. The Cooperative’s data collecting center fetches data from the stations at 15 minute intervals to prevent synchronization problems, and stores them in a flat table, which is then incrementally retrieved by us.

The framework is deployed at LIS (D). This was based on the selection criteria as of [Rademakers and Dirksen 2009]: core functionalities, quality of documentation, market visibility, active community, flexibility and lower development effort, transport/connectivity options, integration with other open source projects and support for implementation through an IDE. We chose Mule ESB [MuleSoft Inc. 2012] to provide the ESB functionalities inside our framework due to its good documentation, active community and good support for implementation.

We deployed a Mule ESB at LIS and used its FTP adapter to retrieve Cooxupé data. We configured a Mule adapter to pool FTP every 15 minutes. After connecting, we filter by file type and translate data inside the incoming stream to our internal message format. Once data are standardized, we deal with event processing detecting patterns and insert data in a database to save historical data.

To provide CEP capabilities we used Esper due to its good available documenta-

[1]The cooperative congregates over 12,000 small farms.

Table 2. Cooxupé Weather stations measurements, within one hour time window.

	Metric name	Description
1	Temp Out	external temperature
2	Hi Temp	highest temperature
3	Low Temp	lowest temperature
4	Out Hum	external humidity
5	Dew Pt.	temperature to which a volume of humid air must be cooled to condense into liquid water
6	Wind Speed	wind speed
7	Wind Dir	wind direction in cardinal points
8	Wind Run	total distance traveled of the traveled wind over a period of time
9	Hi Speed	highest value of wind speed
10	Hi Dir	Most frequent wind direction
11	Wind Chill	effect of wind on temperature humans perceive
12	Heat Index	combines air temperature and relative humidity to determine human-perceived temperature
13	THW Index	Temperature Humidity and Wind Index – calculates apparent temperature
14	THSW Index	Temperature Humidity and Sun Wind Index – calculares apparent temperature
15	Bar	barometric pressure
16	Rain	measured liquid precipitation
17	Rain Rate	amount of accumulated rain over a period of time
18	Solar Rad.	amount of solar radiation
19	Solar Energy	amount of accumulated solar radiation energy over a period of time
20	Hi Solar Rad.	highest measured solar radiation
21	Head D-D	heat amount to keep the structure when the out temperature is one degree low
22	Cool D-D	cool amount to keep the structure when the out temperature is one degree above
23	In Temp	internal (weather station) temperature
24	In Dew	internal (weather station) dew point
25	In Heat	internal (weather station) heat
26	In EMC	internal equilibrium moisture content
27	In Air Density	internal (weather station) air density
28	ET	evapotranspiration

tion, active community and open source license. Esper is used by many customers and is the CEP engine in the Oracle Complex Event Processing [EsperTech Inc. 2012].

One example of event pattern is a basic logical operator with threshold to detect the temperature range between 19 and 22 degrees Celsius (the ideal temperature range to cultivate Arabica coffee):

```
 select temperatureOut from  CooxupeData.win:length(5)
having avg(temperatureOut) > 19.0 and avg(temperatureOut) <
 22.0
```

The second is a basic threshold pattern that detects when it rains. It is important to know the occurrence and the rain rate since water balance affects coffee growth:

```
select rainRate from CooxupeData.win:length(3) having
avg(rainRate) > 0.0
```

Both are written using the Event Processing Language (EPL) from Esper which resemble SQL, but instead of querying tables it queries streams. The EPL written in (i) queries the average outside temperature within a window of length of 5 measurements. If there are data returned, the framework will produce a new event showing that the ideal temperature was reached. In (ii), the query uses the window length of 3 measurements and returns data if the average rainfall in the period is greater than zero.

Within this framework, Cooxupé can provide information on demand or use notifications when something unexpected happens. This can alert farmers about meteorological changes and trigger reactive solutions to avoid damages to coffee production.

5. Related Work

An example of an approach for event processing in a sensor context appears in [Dunkel 2009]. With EPAs that filter, split, aggregate, transform and enrich events. EPAs

are used to propagate more complex and/or abstract events to downstream systems. Sensors are deployed in a physical environment and emit their measurements continuously to EPAs that perform pattern matching and event processing.

Our solution provides another layer of abstraction and adapts Dunkel's proposal using integration capabilities of the ESB before the EPAs can process events or situations. Through this approach it is possible to capture events not only from different devices (*e.g.* sensors), but also from other event sources such as static files, services and databases using the ESB adapters.

Supervisory Control and Data Acquisition (SCADA) [Bailey and Wright 2003] are systems that provide means to acquire data and allow one to perform remote control and event processing. The difference with our proposal is that SCADA focuses primarily in data acquisition and interoperability in the same system. In contrast, we deal with the acquisition of data from different providers and once these data are retrieved provide means to transform, combine and correlate these data.

SensorBUS [Broering et al. 2010] uses the message bus architecture pattern to provide the integration between sensors and services that use the SWE standard specifications. It allows services and sensors to publish messages and to be notified in a publish-subscribe communication. Our proposal is different from SensorBUS because our framework provides not only integration between sensor systems but also between other data providers and uses the CEP to process events.

SAPHE [Churcher and Foley 2010] is a wireless sensor network project in the health care domain that uses SWE specifications and CEP to face access and processing optimization problems. The difference between SAPHE and our approach is that we are interested in dealing not only with sensor data provided by SWE standard services, but we also envisage to use other different data sources.

SStreaMWare [Gurgen et al. 2008] is a service-oriented middleware that aims at dealing with the dinamicity caused by various operations on sensors and their effect on concurrent continuous queries. It uses a specific template data schema and a query language for sensor data called SStreaM. SStreaMWare provides an abstraction to locate, query and aggregate sensor data, but does not deal with other types of data nor treat the events as our approach does.

6. Conclusions

This paper presented our framework to process events from heterogeneous data sources. Our work combines the advantages of using ESB (to deal with integration and extensibility) and CEP (to provide clear and straightforward event processing capabilities). Our case study shows the feasibility of our solution.

There are many possibilities of extensions from the architectural point of view. The first is to develop sensor specific custom transport components to deal with sensor specific (*e.g.* SWE specifications) message formats. Another possibility is to provide easier means to build and deploy new event patterns inside the framework. Yet another involves using semantics in event handling – *e.g.* [Teymourian and Paschke 2010].

Acknowledgements Work partially financed by FAPESP (grant 09/52336-7), the Microsoft Research Fapesp Virtual Institute (eFarms project), CNPq, INCT in Web Science

(CNPq 557.128/2009-9) and CAPES.

References

Babu, S. and Widom, J. (2001). Continuous queries over data streams. *SIGMOD Rec.*, 30(3):109–120.

Bailey, D. and Wright, E. (2003). *Practical SCADA for Industry*. IDC Technologies.

Broering, A., Foerster, T., Jirka, S., and Priess, C. (2010). Sensor bus: an intermediary layer for linking geosensors and the sensor web. In *Proc. COM.Geo '10*, pages 12:1–12:8.

Churcher, G. E. and Foley, J. (2010). Applying complex event processing and extending sensor web enablement to a health care sensor network architecture. In *Sensor Systems and Software*, volume 24 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 1–10. Springer.

Dunkel, J. (2009). On complex event processing for sensor networks. In *Autonomous Decentralized Systems, 2009. ISADS '09. International Symposium on*, pages 1 –6.

EsperTech Inc. (2012). Esper website. http://www.espertech.com/partners/partners.php (Accessed Apr, 2012).

Etzion, O. and Niblett, P. (2011). *Event Processing in Action*. Manning.

Gurgen, L., Roncancio, C., Labbé, C., Bottaro, A., and Olive, V. (2008). Sstreamware: a service oriented middleware for heterogeneous sensor data management. In *Proc. 5th Intl conf. on Pervasive services*, ICPS '08, pages 121–130.

Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional.

Luckham, D. C. (2001). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Luckham, D. C. (2006). What's the Difference Between ESP and CEP? http://www.complexevents.com/2006/08/01/what's-the-difference-between-esp-and-cep/ (Accessed Jun, 2012).

Magid, Y., Sharon, G., Arcushin, S., Ben-Harrush, I., and Rabinovich, E. (2010). Industry experience with the IBM Active Middleware Technology (AMiT) Complex Event Processing engine. In *Proc. 4th ACM DEBS '10*, pages 140–149.

Menge, F. (2007). Enterprise service bus. *Free and Open Source Software Conference*.

MuleSoft Inc. (2012). Mule website. http://www.mulesoft.com/ (Accessed Apr, 2012).

Rademakers, T. and Dirksen, J. (2009). *Open-Source ESBs in Action*. Manning Publications Co., Greenwich, CT, USA.

Teymourian, K. and Paschke, A. (2010). Enabling knowledge-based complex event processing. In *Proc. EDBT/ICDT Workshops*, EDBT '10, pages 37:1–37:7, New York, NY, USA. ACM.