Ontology-Driven Complex Event Processing in Heterogeneous Sensor Networks

Kerry Taylor[1] and Lucas Leidinger[2]

[1] CSIRO ICT Centre
GPO Box 664, Canberra, ACT, 2602, Australia
kerry.taylor@csiro.au
[2] University of Applied Sciences of the Saarland
lucas.leidinger@gmx.de

Abstract. Modern scientific applications of sensor networks are driving the development of technologies to make heterogeneous sensor networks easier to deploy, program and use in multiple application contexts. One key requirement, addressed by this work, is the need for methods to detect events in real time that arise from complex correlations of measurements made by independent sensing devices. Because the mapping of such complex events to direct sensor measurements may be poorly understood, such methods must support experimental and frequent specification of the events of interest. This means that the event specification method must be embedded in the problem domain of the end-user, must support the user to discover observable properties of interest, and must provide automatic and efficient enaction of the specification.

This paper proposes the use of ontologies to specify and recognise complex events that arise as selections and correlations (including temporal correlations) of structured digital messages, typically streamed from multiple sensor networks. Ontologies are used as a basis for the definition of contextualised complex events of interest which are translated to selections and temporal combinations of streamed messages. Supported by description logic reasoning, the event descriptions are translated to the native language of a commercial Complex Event Processor (CEP), and executed under the control of the CEP.

The software is currently deployed for micro-climate monitoring of experimental food crop plants, where precise knowledge and control of growing conditions is needed to map phenotypical traits to the plant genome.

1 Introduction

Sensor networks, especially low-cost wireless sensor networks (WSNs), are rapidly gaining popularity for use in developing scientific knowledge. Scientists are performing dense monitoring of natural environment parameters to learn about matters including faunal distribution and behaviour, biodiversity and biological interactions, air and water quality, micro-climatic conditions, and human impact. In some cases a regular collection of homogenous sensor data is sufficient

G. Antoniou et al. (Eds.): ESWC 2011, Part II, LNCS 6644, pp. 285–299, 2011.

to support subsequent intensive data analysis over a data archive. In other cases, there is a need for real-time data collection, analysis, and active response. This will arise when scientists are unsure about the how to detect the phenomena being investigated, when a human response is necessary to an observed event, or when the recognised occurrence of an event should cause a change in the monitoring behavior or equipment configuration. For example, when turtle eggs begin to hatch on the beach in stormy weather, a scientist should be alerted and cameras should be activated. When an endangered plant begins to flower and pollinators are detected in the locality, a protecting cloche should be removed to enable pollination. When nitrate concentrations at several points in the water course exceed a threshold, the downstream oyster farmers should be warned and additional water quality parameters should be monitored.

In these cases the events may be detected only by recognising a complex correlation of observations made over multiple sensors attached to multiple WSN nodes, and possibly distributed over multiple networks. The sensors may be heterogenous, as may be the sensing nodes that control them. Although in principle is possible to use in-network processing techniques by explicitly programming each device to make observations and to coordinate and correlate those observations with neighbouring nodes, this is very difficult in general and certainly requires advanced programming skills and dedication with current technology. If we add the typical experimental challenge to the mix—that it is not well known *how* to define the desired event in terms of measurable properties, nor even *what* all the interesting events might be over the life of a sensor network deployment, then we can conclude that better tools are needed to offer this capability.

A better tool should enable an experimental scientist to

- discover sensors that could be used to make relevant measurements;
- develop a specification for the event of interest in terms of the available sensing capability;
- reuse measurements that are already being made if possible or to
- program the sensor devices to make the necessary measurements otherwise;
- describe an action to be taken if the interesting event is detected; and to
- easily deploy the specification for efficient runtime processing.

In this paper we propose the use of ontologies as an important part of such a tool. In earlier work we have shown how to effectively program sensor networks and devices by modelling the sensing capability and programming language in an OWL ontology, using the ontology to add contextual concepts for use in the programming task, and using the ontology reasoning capability to validate commands [12]. In this work, we propose that an ontology can capture valuable domain context for sensor capability description and discovery, for description of an interesting event in terms of potential sensor measurements, and for optimising the execution strategy for run-time event detection.

We do not address sensor discovery, but discovery methods based on ontological descriptions abound, for example [11] and [6]. In our work, we rely on the newly developed SSN ontology from the W3C Semantic Sensor Network Incubator Group [13], to take advantage of an emerging community terminology.

For enactment of event detection, we employ one of the many commercial or open source *complex event processors* (CEP) or distributed stream management systems (DSMS). These software systems are capable of binding to input *streams* of real-time structured messages, such as those arising as observation values reported from sensor networks. They can efficiently evaluate queries over those streams that check message content for particular values and for values related to previous messages in the same stream or to messages in other streams. They support temporal and aggregation operators to enable the detection of events that arise as complex correlations of data values in the messages. These systems, too, are notoriously difficult to configure [5] although they commonly offer an SQL-like query language with which to express the desired events. There is no standard language (although they are usually similar to CQL[2]) nor even a widely-used algebra or well-defined semantic model for events.

In our work, we develop an ontology of domain, event, observation and sensor network, and use that ontology to drive a user interface. When the user specifies an event of interest through the interface, our middleware processes the specification and generates configuration commands for a CEP system. The CEP system monitors the selected data streams and generates alerts to be delivered to specified clients when the event occurs. Our contribution is the general method for a *better tool* for scientists as described above, although this paper focuses on the parts that offer development and optimised deployment of the specification of a complex event.

Outline. We begin our description of the method for ontology-driven complex event detection by presenting the overall architecture and the role of the ontology and CEP in section 2. In section 3 we describe the ontology driven user interface, and how it is used to specify complex events over sensor measurements. In section 4 we describe how our semantic middleware processes the specification together with the initial ontology to generate optimised configuration instructions for the CEP middleware. In order to apply the system each sensor network needs an interface developed as described in section 5. After an analysis of related work in section 6 we conclude with a summary and presentation of future work.

2 Architecture - Event Framework

Our *event framework* is software that implements our method for ontology driven complex event processing in heterogeneous sensor networks.

The event framework includes all software to define, process and recognize complex events in sensor networks. The framework is composed of five separate parts, each of which may be replaceable for different applications, although the semantic event middleware is the core component:

User interface to build complex event definitions;
Semantic event middleware to process complex event definitions;
Management module interfaces to program and access different kinds of sensor data sources;

Event ontology containing the model of sensor networks, sensor programs, complex events and the event environment; and

CEP server to perform the complex event detection over multiple heterogeneous sensor network data streams (*Coral8*[1] in our case).

Fig. 1 shows the event framework and illustrates the communication between components. The basic process can be described as follows:

1. The user defines a complex event definition composed from several atomic sensed events within the user interface.
2. The complex event definition is processed and stored in the event ontology.
3. The ontology data is used to program the required sensors.
4. The event detection part for the current event definition is separately saved in an *ExportOntology* and sent to the semantic event middleware.
5. The semantic event middleware transforms the received ontology into CEP streams and queries, written in the CEP-dependent Event Processing Language (EPL).
6. The semantic event middleware sets up the CEP server with created streams and queries, and initiates the event detection process.
7. The CEP server performs the event detection and sends an alert if the specified event has been recognized.

The *EventOntology* is the central part of the Event Framework. It allows the use of reasoning over event information to obtain additional knowledge and to perform semantic optimisations. A clear formalization of the event and measurement environment is necessary to exploit these advantages. Our OWL 2.0 event ontology extends an early form of the SSN ontology of the W3C SSN-XG [13]. It is reported to be in ALCIQ(D) by the ontology editor, Protégé. Along with the sensors, it models the domain of application and the concept and model of the entire event framework. Events, alerts, triggers, streams, locations, instruments, phenomena, sensors and sensor programs are all part of this description. Additional classes and instance data are included to describe relations between single events, time intervals, and to define different kinds of sensor programs. The high-level structure is apparent through its reflection in the user interface given in section 3. The interested reader may find the ontology at `http://research.ict.csiro.au/conferences/ssn/EventOntology_no_imports.owl`.

The CEP server application performs the actual complex event detection. For this, the server receives a stream with configuration information for the event data sources and a query for the complex event detection, both expressed in the CEP's proprietary EPL. The CEP server is also responsible for sending the user defined alert message if an event has been detected.

We now provide more detail on our user interface, our semantic event middleware and our management module interfaces for each source.

[1] The Coral8 CEP-platform.
`http://www.aleri.com/products/aleri-cep/coral8-engine`

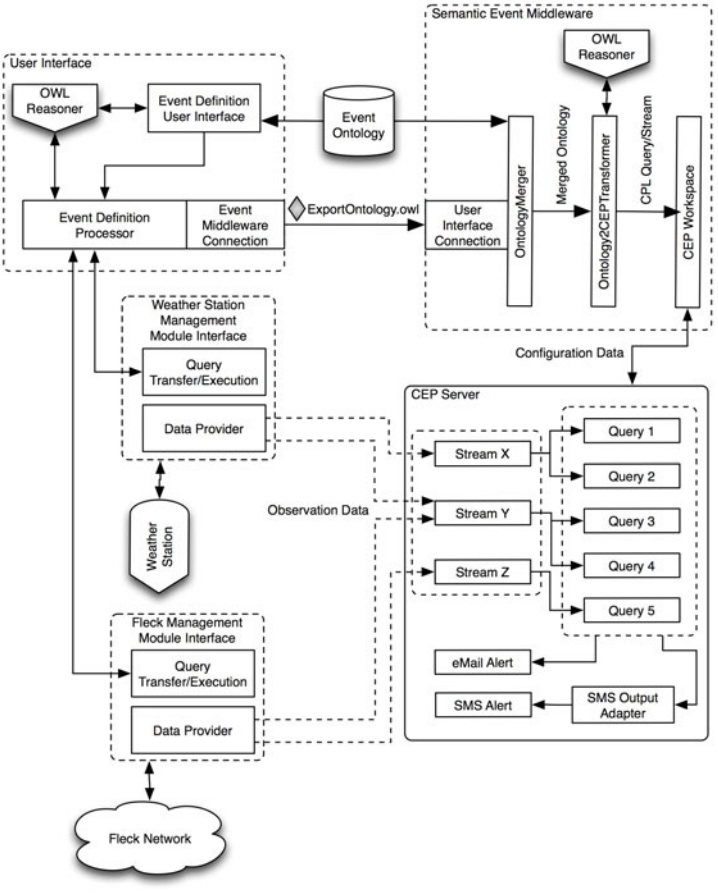

Fig. 1. Event Framework

3 Ontology-Driven User Interface

The user interface is implemented as a plug-in to the popular OWL 2.0 ontology editor *Protégé*. It contains windows to manage new locations and instruments, and to create complex event definitions as shown in see Fig 2. It has the functionality to store entered information as ontology instance data, the capability to perform semantic optimisations, and to program the sensor devices before complex event descriptions are sent for the further processing to the semantic event middleware. The sensor programming capability [12] is not further described.

The ability to provide a user interface which allows one to define events in a logical and expressive way and also to store this definition in an ontology requires splitting the entire complex event definition into parts: *Events*, *Alerts*, *Observations*, and *Triggers*. Each part describes exactly one event property that is able to be stored within an ontology.

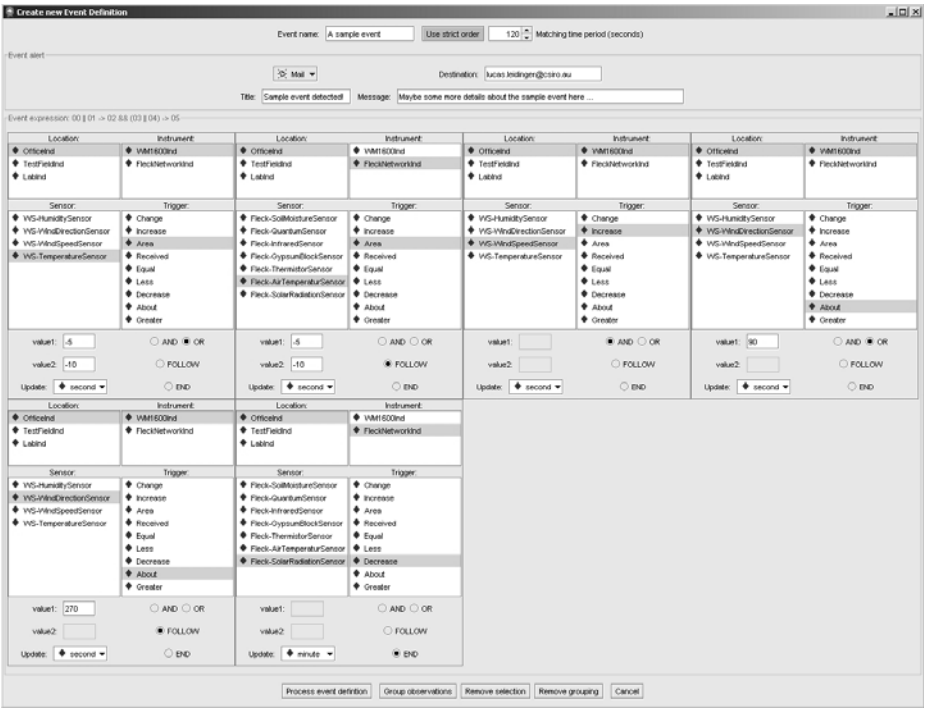

Fig. 2. User Interface

3.1 Events

Every event description consists of two major parts: an alert which will be activated if an event has been recognized, and the definition of the event itself. The definition can be expressive and must be able to represent user-defined complex structures. To achieve this goal, every complex event definition, called an *observation*, is composed of several atomic observations. Each atomic observation can be individually configured in four main steps.

1. Select a sensor from an instrument at a specific location.
2. Choose the update interval for this sensor.
3. Set up a trigger including trigger values.
4. Define the relationship to the following observation.

Defining the relationship to the following observation together with functionality to group sequential observations enables the formation of complex event descriptions. The design of instruments and locations declares that every instrument is located at a specific location. This allows the grouping of all instruments from one location and the selection of instruments based on the location name. The location itself has no finer definition in this implementation, although it

could be useful to add extra information like physical values or the kind of location and an environment description. Both location and instrument descriptions are stored within the ontology. The user interface loads this information dynamically and only displays valid entries within the event definition window.

3.2 Alerts

The user can choose to receive an email or a text message on a mobile phone if an event has been detected. Every email includes standard passages about the event processing supplemented with event specific information such as the latest sensor data and a description of the trigger configuration. The SMS alert is shorter. Alerts are defined within the ontology, together with the relevant EPL code fragments, and dynamically loaded and integrated into the user interface. This easily allows the integration of additional alert types. In future work we will add an alert type to send a control message to external software systems.

3.3 Observations

Complex events are designed within the ontology, so that every complex event contains an observation. An observation itself is used as a generic term for five different kinds that are used to realize the complexity of definitions. Observation operators are used to define the logical *AND*, *OR* and *FOLLOWED BY* relationships between atomic or compound observations. Bracketed grouping can also be represented in observation groups.

Atomic Observation is the description of an atomic event within the entire complex event definition. It contains the information to program the selected sensor and the trigger definition for the event.

Observation Intersection is used to create a logical *AND* (&&) relationship between two observations. Each of these observations can be a single observation, an observation union, an observation sequence or another observation intersection in turn.

Observation Union is the counterpart to the observation intersection. It links two observations by a logical *OR* (||) relationship. Here again, each of observations can be a single observation, an observation intersection, an observation sequence or another observation union.

Observation Sequence is used to create a *FOLLOWED BY* (,) relationship between two observations. *FOLLOWED BY* specifies that the next event has to be recognized chronologically after the previous one. Each of these observations can be a single observation, an observation intersection, an observation union or an observation sequence. This is only available if strict order is used, as described in the next paragraph.

Observation Groups are used to combine multiple single observations. Each group belongs to a certain event and contains all consecutive observations summarized by the user within one parenthetical group.

It is not possible to define *AND* and *OR* relations using the corresponding primitives of the underlying ontology language itself because we also need to use the custom *FOLLOWED BY* operator to express temporal relationships between atomic observations.

The *FOLLOWED BY* relationship is used to define temporal sequences of atomic events meaning that the next event within the event expression must arrive after the previous one. A time period specifies the duration within which individual observations can be triggered in order to be valid and taken into account for the entire complex event detection. Using a strict temporal order allows the defintion of event definitions such as: Send an alert if the temperature at location *A* or at location *B* falls below 0 degrees and then humidity becomes less than 15 percent at location *A* followed by wind speed with more than 130 km/h in combination with a north-easterly wind direction at location *B* within a period of 30 seconds.

An important part of the observation design is that the ontology includes code fragments to generate CEP-platform-specific statements which are used for the run-time event detection.

3.4 Triggers

To create expressive and practical event descriptions, it is necessary to interpret, analyse and filter environmental observations to be able to define which occurrences are interesting for an event. The data source is already described by the sensor program within an atomic observation. In order to be in a position to recognize complex events, it is not enough to simply compare incoming values. It is much more revealing to observe time dependent behaviour and value patterns. For this reason, nine different kinds of *Triggers* were designed.

About examines if the received data is equal to a user defined value. Values within 10 percent tolerance are detected.
Area recognizes all readings in the interval between two given values.
Change monitors if the current reading changes with respect to the average value of the previous two readings.
Decrease is used to detect if the latest value is lower than the average of two preceding readings.
Equal simply checks if the current value equals a value defined by the user.
Greater triggers if the received value is greater than a user defined value.
Increase is the opposite of Decrease and observes if the latest value is greater than the average of two preceding readings.
Less triggers if the received reading is smaller than a user defined value.
Received recognizes if data from a sensor has been received, without further qualification.

Like the observation concept, the trigger specification within the ontology contains CEP-platform specific code fragments which are used to generate queries

for the event detection. The trigger implementation focuses on demonstration purposes and is quite elementary. The calculations of limiting values as well as the limiting values itself were chosen arbitrarily to obtain a satisfactory behaviour in the targeted environment. This design is sufficient to demonstrate the technical feasibility of defining complex triggers in an ontology and to transform them into CEP statements.

4 Semantic Event Middleware

All complex event descriptions in the form of an ontology are forwarded from the user interface to the semantic event middleware. The semantic event middleware uses the complex event descriptions to generate streams and queries for the CEP server application to enact the event detection. Subsequently, the CEP server is connected, streams and queries are registered, and the event detection process started.

4.1 Communication with the User Interface

Within the semantic event middleware, the same ontology as in the user interface*EventOntology*, is used. New complex event descriptions generated in the user interface are received by a network connection, so that the event definition and the event transformation can be performed independently. Only ontology fragments for new complex event descriptions are exchanged; they are merged with the local *EventOntology* on receipt.

4.2 Transforming Ontology Data into EPL Statements

Through the decomposition of the complex event definitions into individual parts, describing exactly one property, it is possible to extract the entire event description from the ontology and make it usable to generate CEP program code. These CEP programs are composed of streams and queries. *Streams* are used to create connections to event data sources and to filter the required data for the event detection. *Queries* describe the actual event detection including all observed sensors, triggers and relationships between atomic observations, groups and the temporal order of observations.

At this point, code fragments, which are stored as datatype property values in the event ontology are used to form EPL statements. Furthermore, class and instance names from the ontology are used to generate names for variables in the CEP code. Together, this supports the generation of large parts of the CEP code automatically and independently of the particular EPL. Storing code fragments not only for one CEP implementation but for multiple CEP-platforms in the ontology would additionally allow the choice of different CEP implementations. However, as the grammar of the EPL needs to be reflected in the translation of the *Ontology2CEPTransformer* (fig. 1), alternative implementations of this module may be necessary for CEP platforms employing a very different language structure.

Fig. 3 illustrates this usage of ontology data to create Coral8 EPL program code. The shown ontology data and the code snippet correspond to a complex event definition example which recognizes temperature between -20 and -10 degrees, amongst other observations. All shaded statements are generated from the description of sensor programs from the ontology. For example, the sensor program individual "program_0_1283418466826Ind" has the object properties "WS-TemperatureSensor" and "WM1600". Both are used as variable names to describe input and output streams as well as CEP windows that include the definition of which data has to be filtered. The black highlighted expression within the "where" clause is also created automatically. The template clause, "trigCmd", for the *Area* trigger, "column > value1 AND column < value2", is used as basis for the "where" clause. This information is stored in the ontology as a part of the definition of the *Area* trigger class, and so, through a reasoner, is also a dataproperty value of the "trigger_0_1283418466826Ind" individual of that class. Other data property values instantiated through the user interface provide the user defined thresholds "hasValue1" and "hasValue2" to replace the terms "value1" and "value2" inside the template clause. The template string "column" is replaced by the corresponding variable name generated by using the description of sensor program "program_0_1283418466826Ind".

4.3 Semantic Optimisation of Streams

Reasoning is used to improve the efficiency of streams within the semantic event middleware. The goal is to reuse existing information to save run-time resources and to reduce the amount of data which must be transferred between the data sources and the event processing application. This may be especially important when multiple users are being served.

Since CEP platforms and the *Semantic Event Middleware* are able to monitor several complex events at the same time, it is possible to re-use some streams already configured in the CEP for previously specified complex events. When a stream definition is created through this software through commands to the CEP, it is also defined in the *EventOntology* as an instance of the stream class with associated property instances. For example, if streams which provide temperature and wind speed from location *plot21north* and wind direction from location *plot23central* have previously been established, then a new query which only requires wind speed from *plot21north* and wind direction from location *plot23central* can reuse the two existing streams. The usable pre-existing streams are retrievable by a reasoner, being the individual members of the class described as a stream which measures wind speed at *plot21north* and the individual members of the class described as a stream which measures wind direction from location *plot23central*.

5 Management Module Interfaces

In order that different sources can be used for the event processing, the module for specifying and providing event data streams is abstracted. The usage of a

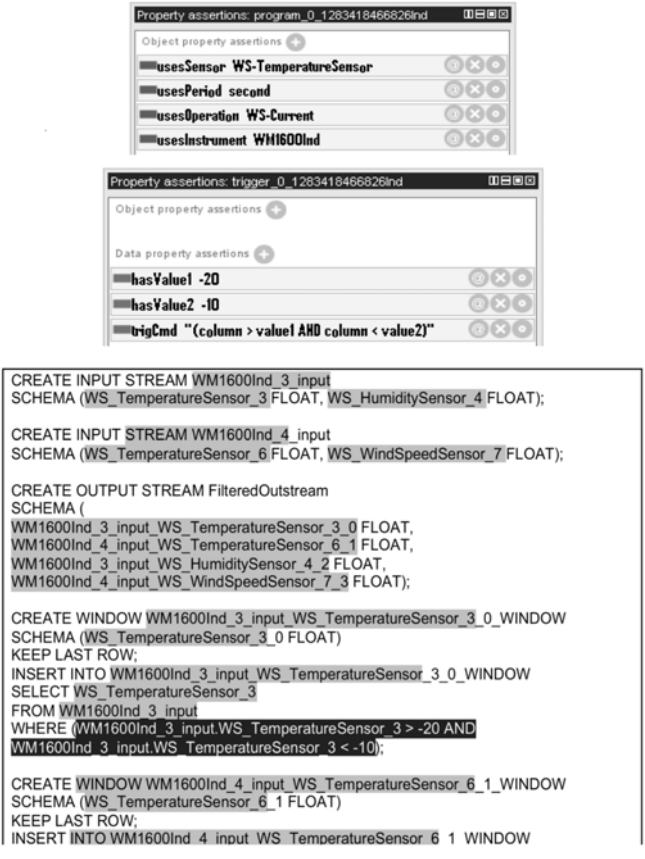

Fig. 3. Using ontology information to generate Coral8 CEP statements

management module interface allows the implementation of an independent solution for each specific kind of instrument, sensor network or event source. for our test deployment have developed two instances of the management module interface, one to connect a *WeatherMaster1600*[2] weather station instrument and the other to interface with a relational database archive. Each additional management module interface would be accessed, integrated and implemented in the same manner. It would be necessary to update the user interface source code to use a new management module interface as well as to extend the model of the *EventOntology* with new instrument specific capabilities.

Most parts of the weather station management module interface are reused from the related project [12]. The software allows high-level programming for the weather station instrument and access to measured data.

[2] Environdata WeatherMaster1600.
`http://www.environdata.com.au/weathermaster1600`

6 Related Work

Research over the past twenty years has produced a large number of research prototypes and commercial products to continuously monitor data streams for detection of events arising as complex correlations of source data values. These may be known as distributed stream management systems, complex event processors, or sensor data management systems. Some are directed at scientific users and incorporate grid computing elements; others are optimised for high message throughput rates; and others again are closely integrated with the sensor networks that generate the data and can instigate collection of data from the sensors on demand. Some of the best known are Aurora/Borealis[1], TinyDB [10], Caldera[9], and Esper [4].

In concurrent work, [3] extends the standard RDF query language SPARQL to provide a SPARQL interface to querying streaming data sources, together with a temporally extended RDF response. One aim of that work was to make streaming sensor data available within the linked open data framework, hence the choice of an RDF/SPARQL model. However, like in our work, the queries are mapped to to a native stream processing language for run-time execution. In that case a more traditional query translation approach is used with an intermediate algebraic representation, and reverse translation of query answers.

Our event framework does not offer a query language interface directly but we offer an exploratory GUI that can be understood as defining an interesting event rather than querying for instances of the event. However, because an event specification in our approach is simply an ontology fragment, other query-language like interfaces or API could be readily designed. A query language based on description logic conjunctive queries over the classes and properties of an event definition would be a more natural match for our work than extended SPARQL.

Although we do not offer a linked open data solution, we rely on the more expressive OWL ontology language to provide design-time contextualisation of the sensor data and optimisation, but with no run-time processing overhead. Our approach provides a more direct translation path to the underlying EPL, so is likely to allow more expressive queries (where modelled in the ontology) and possibly also more compact and efficient EPL code. For example, our system straightforwardly offers complex events defined by integrating multiple sensor data streams, whereas that capability is set down for future work in [3].

In [6], a SensorMashup platform is described that offers a visual composer for sensor data streams. Data sources and intermediate analytical tools are described by reference to an ontology, enabling an integrated discovery mechanism for such sources. A SPARQL endpoint is offered to query both sensor descriptions (in the conventional manner) and also individual sensor data streams. Like our event framework, a third-party DSMS is used to manage the raw sensor data streams. A SPARQL query is translated to an SQL-like continuous query over the streams to be handled by the DSMS, but the usual essential windowing and aggregation functions of a DSMS (such as "average") cannot be used as there is no SPARQL counterpart. The most important difference to our

framework is in the approach to queries requiring correlations over multiple independent streams. In SensorMashup such queries are written, mapped and processed separately for each input stream. Although the visual composer supports the explicit combination of streams, it appears that this means feeding the streams as inputs to subsequent processing steps embedded in arbitrary software services, so correlation over the streams to recognise combined events must be done outside the DSMS. In our approach a single declarative query over multiple streams is mapped directly to a declarative query in the language of the CEP, therefore enabling the CEP to perform run-time optimisation of the processing for which it is designed. On the other hand, other than the basic operators embedded in our CEP, our current approach does not support integration of analytical tools.

7 Future Work and Conclusion

We have shown that it is possible to offer a sophisticated, domain-contextualised service for complex event processing. The ontology can be used to specify events that comprise complex correlations of observations over multiple sensor data streams, and to specify an action to be taken when events occur. The ontology is used only for event definition and optimisation and is compiled out of the specification for run time processing, where a third party high-throughput complex event processing engine is used. This approach offers the advantages of semantic descriptions but without the cost of semantic interpretation at run time, permitting choice of the best CEP or DSMS tool for stream processing.

Most such processors can be used, provided they offer a query language interface. Because the ontology itself holds command language fragments, and most of the command languages have a similar syntax, the adaptation to a different event processor may require no more than replacement of the fragments in the ontology. Some alteration to the code managing the CEP interface might also be required. For broader sensor network applications, it will be necessary to extend the complexity of the aggregation and relationship operators over stream values beyond those that are available natively in the CEPs. We plan to incorporate a statistical analysis component, and possibly also a textual analysis component to support this. We expect this will slow the run-time processing but it will offer added functionality in relatively low throughput applications.

We have described our work in terms of the Protégé plug-in user interface to construct event specifications. Although our user interface is ontology-driven, we expect that the capability it offers would, in some cases, be better built into other tools more specifically customised to the domain of application. Recalling that the task of the user interface is only to supply an ontology fragment that drives the downstream processing, this should not be difficult.

We have integrated the work reported in this paper with our earlier work for ontology-driven sensor network programming[12]. Within one ontology-driven interface a user can both program the sensor networks and specify actions to be

taken when complex events arising from the sensed data are detected. Furthermore, a user can specify a complex event of interest, within the capability of the available sensor networks, and if the necessary phenomena are not currently being monitored, the system will automatically and transparently program those various sensor networks to monitor the necessary phenomena.

Our work is currently deployed on our Phenonet network for agricultural monitoring installed near Leeton, NSW, Australia. Although a fairly small deployment, the architecture is highly heterogeneous. The network includes several Fleck devices[3] of a WSN with sensors for soil moisture, leaf temperature, and soil moisture. There is a separate Fleck WSN with the node directly connected to a Vaisala Automatic Weather Station [4], a solar radiation sensor, and a photosynthetically active radiation sensor. Another independent wireless network of Hussat data loggers[5] with soil temperature profile and soil moisture profile sensors is also present. Finally there is an independent IP-connected Environdata automatic weather station. Currently, stream data arising from Fleck and Hussat nodes is retrieved by polling a database archive and automated programming of the nodes through the event framework is not possible. The programming capability and live stream feed for these sources will be available shortly, taking advantage of code optimisation work [7] in a service architecture [8].

One event that is particularly important in this domain is the occurrence of frost. We will investigate including frequent weather reports as a source of streaming data together with our sensors in the field. We will use the system to develop an adequate recognition of a frost occurrence and connect the event notification to the control system for infrastructure that can protect the experimental crop in the field.

The strength and novelty of this work lies in its use of ontologies and reasoning. We have shown how a scientist can develop a specification for an event of interest in terms of the available sensing capability, reusing measurements that are already being made. We have shown how a scientist can describe an action to be taken if the interesting event is detected, and can easily deploy the specification for efficient runtime processing. Because of the ontological component, the work can also be used together with semantic discovery techniques and also semantic sensor network programming techniques to offer a complete solution for user-driven scientific and experimental reuse of heterogenous sensor networks.

Acknowledgement. The authors thank the members of the CSIRO semantic sensor networks team, Peter Lamb, Doug Palmer, Laurent Lefort, Leakha Henry and Michael Compton, and the CSIRO Phenomics scientists, Xavier Sirault and

[3] Powercom Fleck. See
`http://www.powercomgroup.com/Latest_News_Stories/Fleck_long_range_wireless_sensing_and_control.shtml`

[4] Vaisala WM30. See
`http://www.vaisala.com/files/WM30_Brochure_in_English.pdf`
for the data sheet

[5] Hussat wireless microloggers. See
`http://hussat.com.au/`

David Deery. This work was conducted using the Protégé resource, which is supported by grant LM007885 from the United States National Library of Medicine.

References

1. Abadi, D.J., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.B.: Aurora: a new model and architecture for data stream management. VLDB Journal 12(2), 120–139 (2003)
2. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: Semantic foundations and query execution. Very Large Database (VLDB) Journal 14 (2005)
3. Calbimonte, J.-P., Corcho, O., Gray, A.J.G.: Enabling ontology-based access to streaming data sources. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 96–111. Springer, Heidelberg (2010)
4. Esper–Complex Event Processing. Espertech event stream intelligence (December 2010), `http://esper.codehaus.org/`
5. Hinze, A., Sachs, K., Buchmann, A.: Event-based applications and enabling technologies. In: DEBS 2009: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, pp. 1–15. ACM, New York (2009)
6. Le-Phuoc, D., Hauswirth, M.: Linked open data in sensor data mashups. In: Taylor, K., Ayyagari, A., De Roure, D. (eds.) Proceedings of the 2nd International Workshop on Semantic Sensor Networks, SSN 2009, Washington DC, USA, October 2009, vol. 522, pp. 1–16 (2009) CEUR workshop proceedings
7. Li, L., Taylor, k.: Generating an efficient sensor network program by partial deduction. In: Zhang, B.-T., Orgun, M.A. (eds.) PRICAI 2010. LNCS, vol. 6230, pp. 134–145. Springer, Heidelberg (2010)
8. Li, L., Taylor, K.: A framework for semantic sensor network services. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 347–361. Springer, Heidelberg (2008)
9. Liu, Y., Vijayakumar, N., Plale, B.: Stream processing in data-driven computational science. In: Proc. 7th IEEE/ACM International Conference on Grid Computing, GRID 2006, pp. 160–167. IEEE, Washington, DC, USA (2006)
10. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30, 122–173 (2005)
11. Sirin, E., Parsia, B., Hendler, J.: Filtering and selecting semantic web services with interactive composition techniques. IEEE Intelligent Systems 19, 42–49 (2004)
12. Taylor, K., Penkala, P.: Using explicit semantic representations for user programming of sensor devices. In: Advances in Ontologies: Proceedings of the Australasian Ontology Workshop Conferences in Research and Practice in Information Technology, Melbourne, Australia, December 1. CRPIT, vol. 112, Australasian Computer Society (2009)
13. W3C SSN-XG members. SSN: Semantic sensor network ontology (December 2010), `http://purl.oclc.org/NET/ssnx/ssn`