EQL-CE: An Event Query Language for Connected Environments

Elio Mansour, Richard Chbeir, Philippe Arnould

EQL-CE: An Event Query Language for Connected Environments

Elio Mansour
Univ Pau & Pays Adour
E2S UPPA, LIUPPA
Mont-de-Marsan, 40000, France
elio.mansour@univ-pau.fr

Richard Chbeir
Univ Pau & Pays Adour
E2S UPPA, LIUPPA
Anglet, 64600, France
richard.chbeir@univ-pau.fr

Philippe Arnould
Univ Pau & Pays Adour
E2S UPPA, LIUPPA
Mont-de-Marsan, 40000, France
philippe.arnould@univ-pau.fr

ABSTRACT

Recent advances in sensor technology and information processing have allowed connected environments to impact various application domains. In order to detect events in these environments, existing works rely on the sensed data. However, these works are not re-usable since they statically define the targeted events (i.e., the definitions are hard to modify when needed). Here, we present a generic framework for event detection composed of (i) a representation of the environment; (ii) an event detection mechanism; and (iii) an Event Query Language (EQL) for user/framework interaction. This paper focuses on detailing the EQL which allows the definition of the data model components, handles instances of each component, protects the security/privacy of data/users, and defines/detects events. We also propose a query optimizer in order to handle the dynamicity of the environment and spatial/temporal constraints. We finally illustrate the EQL and conclude the paper with some future works.

KEYWORDS

Event Query Language, Internet of Things, Sensor Networks

1 INTRODUCTION

Recent advances in the fields of Information & Communication Technologies (ICT), Big Data, Sensing Technologies, and the Internet of Things (IoT) have paved the way for the rise of smart connected environments. These environments are defined as infrastructures that host a network of sensors capable of providing data that can be later mined and processed using advanced techniques, for high level applications. Hence, Sensor Networks (SN) are currently impacting numerous domains (e.g., medical, environmental, cities, buildings). This allowed a plethora of sensor-based applications such as monitoring a patient's health [20], detecting fires in the wilderness [24], monitoring pollution levels or traffic congestion in a city[15], and optimizing energy consumption/occupants' comfort in buildings[1, 8, 14, 19, 23, 25]. Even though these applications have different objectives, they all rely on sensed data from the environment in order to detect specific events (e.g., a stroke for a patient, a volcanic eruption, a storm, polluted air in a city, temperature rising in an office). Therefore, these applications share the following needs: (i) representing the infrastructure and the sensor network of the connected environment; (ii) defining and detecting the targeted events; and (iii) protecting the security of the sensed data and the privacy of the users in the environment (e.g., protecting patients' medical records). In the aforementioned works, the authors do not emphasize on the environment's representation and define the events statically. They also proposed event detection mechanisms that perfectly fit the description of the targeted events. This is constraining since these works are not re-usable in different contexts. Event Query Languages (EQL) have been proposed to overcome this issue. Users express their needs through EQLs by defining the structure of the targeted events. However, existing languages [2, 3, 6, 7, 9–11] focus mainly on the event descriptions and do not consider other environment components (e.g., infrastructure, sensor network, application domain). They share the following limitations:

(1) Lack of *considered components*. It is important that the EQL allows the definition of the entire connected environment. This includes components related to the environment itself, its sensor network, the targeted events, and the application domain.
(2) Lack of *considered functionality*. It is important that the EQL (i) allows the definition of components (e.g., buildings, sensors, data, events); (ii) allows the manipulation of component instances (e.g., inserting new instances, updating, deleting, selecting them); and (iii) protects the security/privacy of data/users.
(3) Lack of *re-usability*. It is important that the EQL remains generic and independent from any technological constraints or underlying infrastructure. Some languages heavily rely on a specific syntax or data model (e.g., SQL-based, or SPARQL-based) and this limits their re-usability in different contexts.

In order to consider the dynamicity of connected environments and spatial/temporal distributions, we consider the following limitations as well. First, the difficulty in *handling dynamic environments*. Since sensors might breakdown, or mobile sensors could change locations or even enter/exit the network, sensors/observations that are needed for a previously defined event might become unavailable. Therefore, it is important that the EQL allows query re-writing in order to update obsolete event definitions. This entails replacing missing sensors by others capable of providing the required data or replacing missing observations with others that fit the event definition. Second, the lack of *spatial distribution* of sensors. Since the sensors' locations impact event detection, the EQL should allow users to define spatial distributions of the sensors over the infrastructure in order to better detect the targeted events. This entails specifying where each sensor should be located or how they should be distributed over the space (e.g., nearest sensors to a point of interest, sensors within a range of a point of interest, sensors that fit a mathematical distribution around a point of interest). Finally, the lack of *temporal distribution* of sensor observations. Since sensors provide observations at specific rates, one could end up with either: (i) big volumes of unnecessary data (if the rate is too quick); or (ii)

undetected events (if the rate is too slow). Therefore, it is important to have an EQL that allows the adjustment of the temporal distribution of sensor observations based on events' needs/requirements. This entails specifying which sensor observations/sensing rates are considered for a specific event, or selecting a temporal distribution of these observations (e.g., the closest observations to a certain point in time, all observations within a temporal range, distributed sensing rates).

Many other challenges emerge when considering an EQL for connected environments (e.g., handling big volumes of data, continuous heterogeneous data streams). However, in this paper, we focus mainly on the aforementioned limitations. Hence, we propose here an EQL specifically designed for connected environments and partitioned into three layers: conceptual, logical, and physical. It allows (i) the composition of high level generic queries that can be parsed into various data model specific languages (re-usability); and (ii) full coverage of components and functionality (we will detail security related tasks in a dedicated future work). We also propose a query optimizer module that will handle spatial/temporal distributions and query re-writing in order to redefine components that need to evolve when handling the environment's dynamicity (the optimizer will be fully detailed in a separate work). Our proposal, denoted EQL-CE, is part of a global framework for event detection in connected environments which we will also present in this work.

The remainder of the paper is organized as follows. Section 2 presents a scenario that motivates our proposal. Section 3 evaluates existing approaches. Section 4 presents our event detection framework and details EQL-CE. An illustration example is presented in Section 5. Finally, Section 6 concludes the paper and discusses future research directions.

2 MOTIVATING SCENARIO

In order to motivate our proposal, consider the following scenario that illustrates a smart mall. This is a simplified example that illustrates the setup, the needs, and motivations behind our proposal. Of course, it does not summarize all needs found in a connected environment/event detection application scenario. Figure 1 details the infrastructure's location map, and individual locations (i.e., shops and open areas). The mall is equipped with a hybrid sensor network having static/mobile sensors, single sensor nodes/multi-sensor devices capable of monitoring the environment and producing scalar/-multimedia observations (e.g., temperature, video). A manager uses an Event Query Language (EQL) in order to define/detect interesting events that occur within the mall's premises. Although this seems enough to manage the smart mall, many improvements can still be integrated:

- Need 1: Modeling the environment and its sensor network. Before defining and detecting events, a mall manager needs to represent the smart mall using the EQL. This includes defining the infrastructure (i.e., the mall), the locations (e.g., shops), and their spatial relations. Then, the manager needs to define the sensor network that is hosted in the mall. This entails modeling the available sensors (e.g., temperature, humidity), their deployment locations, the data they sense and so on. Once all component structures are defined, the mall manager needs to use the EQL to create instances of each component (e.g., temperature sensor in food court). This is currently not possible since the EQL used in the example only defines events.
- Need 2: Measuring the average temperature in the grocery store (for food storage concerns). The mall manager uses the existing EQL to define the targeted event (i.e., the average temperature in the grocery store). The EQL allows the manager to consider all sensors within the area of interest. However, Figure 2.a shows that the sensors are not evenly distributed in the store (most are located in the upper left corner). Hence, considering all sensors and calculating the average will produce a biased temperature measure that does not reflect the reality of the situation. This can be solved by allowing the manager to define a specific distribution of sensors over the space (e.g., even distribution, only considering sensors within a range of the center of the store). The current setup is limited since it does not allow the definition of spatial distributions of sensors.
- Need 3: Minimize data overload/missed events. Currently, the manager can use the EQL to define one sensing rate for all sensors or sensor types (e.g., temperature, humidity). This is constraining since (i) a quick sensing rate overloads the system with big volumes of unwanted/unnecessary data; and (ii) a slow sensing rate could lead to missing events that began and ended in a short time lapse. Therefore, the temporal distribution of sensor observations (i.e., a start time, a specific rate, a stop time) should be based on the event definition and therefore considered/handled in the event queries (e.g., selecting the closest observations to a time of interest, considering different sensing rates from various sensors at once). The EQL used by the mall manager does not allow such customization of temporal constraints (cf. Figure 2.b).
- Need 4: Detecting a fire event in Shop 1. The mall manager defines this fire event using the EQL. His/Her definition relies on the smoke, humidity, and CO_2 sensors located in Shop 1. However, what if the smoke sensor broke down ? Or what if the mobile device that he/she was depending on left the shop ? Then, the previously defined event query will become obsolete since there are no more smoke observations coming from shop 1, and there is no way of changing the event definition. Hence, query re-writing is necessary in order to update the definition: (i) by replacing the smoke sensor by another capable of providing the same data (e.g.,

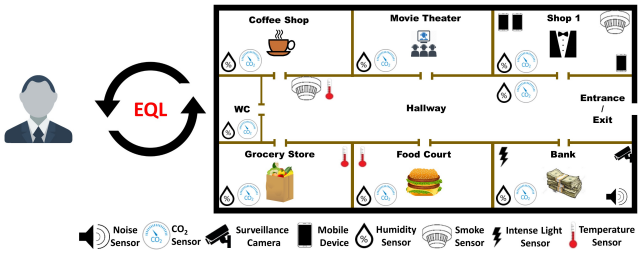
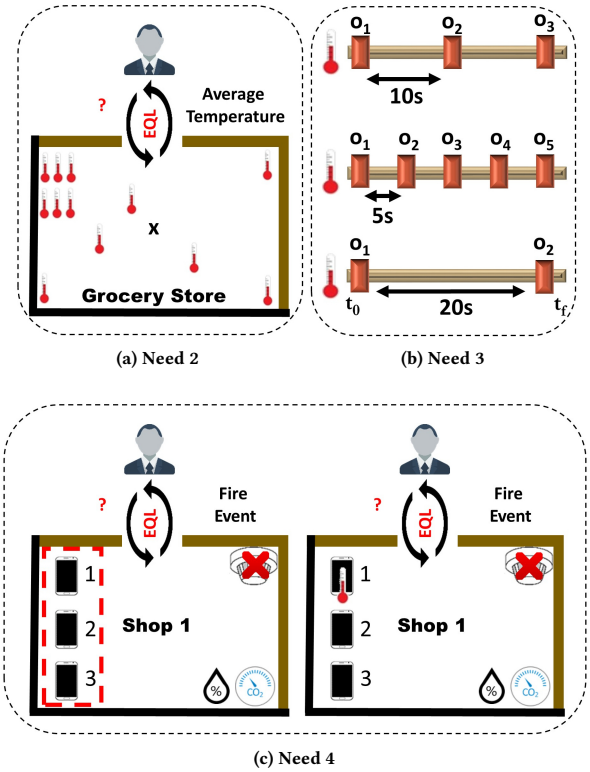


Figure 1: The Smart Mall

mobile device 1 - cf. Figure 2.c.left); or (ii) by replacing the event describing feature smoke by another (e.g., temperature from mobile device 1 if no other sensors can provide smoke observations - cf. Figure 2.c.right). The current EQL is limited since it does not allow such re-writing.



(a) Need 2 (b) Need 3

(c) Need 4

Figure 2: Spatial Distribution (a) | Temporal Distribution (b) | Query Re-writing (c)

In order to address the aforementioned needs, the EQL should provide a means for defining the structure of various components related to the environment, sensor network, targeted events, and application domain. Moreover, the EQL should not be limited to defining components. Its functionality should extend to managing instances of these components, and protecting the security/privacy of the data/users (cf. Need 1). In addition, customizing the sensors' spatial distribution over the infrastructure/environment based on event requirements is required (cf. Need 2). This benefits the event detection since it provides the user with the ability to customize the setup in the way that he/she believes is optimal. The same is also applied for temporal distribution of sensor observations. The EQL should allow the user to select specific observations, or a set of distributed observations in time (e.g., considering different sensing rates, temporal distance to a point in time) when defining the event (cf. Need 3). Finally, the EQL should allow re-writing queries (cf. Need 4) to handle the dynamicity of the connected environment. This is especially beneficial when faults or sensor breakdowns/mobility can render some event definitions obsolete.

However, when considering various components, functionality, data distribution (e.g., spatial, temporal), and query re-writing the following challenges emerge:

- Challenge 1: How to model components and inter-component relations? How to establish ties between the different connected environment elements (i.e., environment, sensor network, events, and application domain)?
- Challenge 2: How to define different query types to cover all the required functionality?
- Challenge 3: How to establish a generic query syntax that can be re-used regardless of the underlying infrastructure (e.g., in a traditional database or in an ontology data model)?
- Challenge 4: How to integrate variables that specify spatial/temporal distributions in the query syntax ? How to propose different types of distribution queries ?
- Challenge 5: How to enable query re-writing upon user request ? How to replace missing sensors/event describing features when re-writing a query ?

Therefore, we propose here a high-level generic event query language, denoted EQL-CE, capable of covering all components. Our covered functionality are partitioned into three main categories for component definition, manipulation of component instances, and data protection (to be discussed in a future work). We also propose a query optimizer that handles query re-writing and spatial/temporal distribution functions. In this paper, we present the optimizer but leave the details of the query re-writing and distribution functions to a separate dedicated work.

3 RELATED WORK

In this section, we review existing works on Event Query Languages (EQL). We propose the following criteria based on the challenges and limitations discussed in Section 2:

- Criterion 1. Component/Functionality Coverage: Denoting if the EQL covers (i) the entire components that constitute a connected environment (i.e., environment, sensor network, application domain, and event related components); and (ii) the entire set of functionality needed for the definition of components, the manipulation of their instances, and protection of the data/user security and privacy (cf. Need 1).
- Criterion 2. Re-usability: Indicating if the EQL is generic and technology independent in order to re-use it in various setups with different underlying infrastructures (e.g., traditional database, ontology). It is beneficial to have a high level, generic, and declarative EQL that can be parsed into data-model specific languages (instances). This facilitates its integration in various contexts.
- Criterion 3. Spatial/Temporal Distributions: Specifying if the EQL allows (i) spatial distribution queries (e.g., selecting sensors that are distributed based on a mathematical law, within a specific range, or near a point of interest); and (ii) temporal distribution queries (e.g., selecting sensor observations that are closest to a point in time that have various sensing rates). This is important for the definition of specific events where such level of detail is required (cf. Needs 2 and 3).
- Criterion 4. Handing Environment Dynamicity: Stating if the EQL provides the means to modify the structure of previously

defined components (e.g., events) in order to keep up with environment changes. This is useful in a dynamic setup, where sensor mobility causes gain/loss of data in certain areas of the environment (cf. Need 4).

We group the existing works into three main categories: (i) conceptual languages (e.g., Event-Condition-Action languages) ; (ii) logical languages; and physical languages (e.g., SQL/SPARQL-based languages). We compare in the following some works from each category (we do not detail here every existing event query language for the sake of brevity).

3.1 Conceptual Languages

This category of languages includes Event Condition Action (ECA) languages that allow the declaration of three event attributes: (i) an event name or label; (ii) a set of conditions (the pattern) that best define the event; and (iii) the set of actions that should be triggered once the event is detected. In [9], the authors propose an intuitive event query language denoted SNOOP. They follow the ECA model when defining event structures. They integrate operators for inter-condition relations (e.g., conjunction, dis-junction, and sequence) and represent repetitive events through the usage of the periodic/aperiodic operators. In [6], the authors propose a language denoted CeDR. In comparison with SNOOP, CeDR adds a WHERE clause for filtering statements and has a wider range of operators. Therefore, CeDR is considered more expressive in terms of event pattern description. CeDR also includes an event lifetime operator and a detection window operator. The authors in [11] propose an event query language for data streams called SaSE. They include the WITHIN and RETURN statements to respectively declare sliding time windows and the required output. SaSE also allows event pattern operators (similar to CeDR) in a WHERE clause.

Discussion: The aforementioned works are intuitive, practical, and allow various composition operators for event definition. Their syntax is also independent from specific data models (e.g., SQL or SPARQL). However, they all suffer from the same limitations. None of them covers the environment or sensor network definition in their queries (cf. Criterion 1 - Component Coverage). They mainly focus on the definition and retrieval of events while neglecting other tasks such as updating definitions or inserting data (cf. Criterion 1 - Functionality Coverage). They also do not consider spatial/temporal distributions (cf. Criterion 3).

3.2 Logical Languages

This category includes works that define events in logic style formulas. To give a few examples, consider ETALIS[3]. This EQL describes events as rules. The authors propose a set of temporal relations and composition operators to define the event patterns. The syntax of the rules is independent of any underlying data model. XChangeEQ[7] is another logical language. The authors allow the following features in its queries: (i) data-related operations such as variable bindings and conditions containing arithmetic operations; (ii) event composition operators such as conjunction, dis-junction, and order; (iii) temporal and causal relations between events in the queries; and (iv) event accumulation, for instance aggregating data from previous events to discover new ones.

Discussion: The aforementioned languages are re-usable in different contexts since their syntax, a logical rule-based notation, is independent of specific data models (cf. Criterion 2). They also cover the majority of temporal and composition operators. However, they do not cover spatial/temporal distributions (cf. Criterion 3). These languages have not fully detailed query re-writing (cf. Criterion 4), and they mainly focus on the events. They cannot be used to define and manage the environment and sensor network components (cf. Criterion 1).

3.3 Physical Languages

This category includes data model specific languages. We detail here languages that were specifically designed for either relational database or linked data management systems. Therefore, the following EQLs are either inspired from or directly extend SQL/SPARQL. ESPER[10] is an implementation for event detection in database systems. The authors proposed an SQL-like syntax for event processing. Therefore, known operators such as CREATE, SELECT, INSERT, UPDATE, and DELETE are available for event definition and detection. ESPER also includes temporal operators and a specific statement for event definition (i.e., the pattern). In addition to the aforementioned advantages, this language has a fast learning curve since it is highly similar to traditional SQL. CQL[4] is another language that can be used for event definition/retrieval. CQL extends SQL by emphasizing on continuous data streams/queries. The authors add temporal operators, sliding windows, and window parameters to better handle continuous data. Many languages extend SPARQL for linked data management systems. For instance, C-SPARQL[5] extends SPARQL to consider stream data in the queries. To do so, the authors integrate sliding time windows. SPARQL-ST[17] extends SPARQL by adding operators for spatial/temporal queries. This covers the definition and manipulation of spatial shapes and temporal entities. Finally, EP-SPARQL[2] integrates event processing operators (e.g., sequence) into the SPARQL syntax. This work allows the definition of simple and complex event patterns in a linked data management system.

Discussion: The aforementioned works are all user friendly since they extend known languages. They cover definition and manipulation queries for various components or entities (cf. Criterion 1). They also provide a basis for spatial/temporal operators and query re-writing. However, distribution queries are not considered (cf. Criterion 3) and their high reliance on a specific data model syntax (SQL or SPARQL) limits their re-usability in different systems (cf. Criterion 2). For instance, EP-SPARQL cannot be used in a relational database infrastructure.

To conclude this section, none of the mentioned works fully considers our entire list of criteria. Therefore, we propose in the following section the Event Query Language for connected environments (EQL-CE). Our proposal has three layers (conceptual, logical, and physical). It ensures re-usability, handles dynamic environments, fully covers the components/required functionality, and integrates spatial/temporal distribution variables in its queries.

4 EQL-CE: AN EQL FOR CONNECTED ENVIRONMENTS

In order to highlight the usage of EQL-CE, we present here an overview of our framework for event detection in connected environments. This framework includes the following modules: (i) a data model representing the connected environment; (ii) an event Virtual Machine (eVM) for event detection; and (iii) an event query language for user/framework interaction. We start by briefly describing these modules. Then, we detail our proposed event query language for connected environments, denoted EQL-CE.

4.1 Event Detection Framework

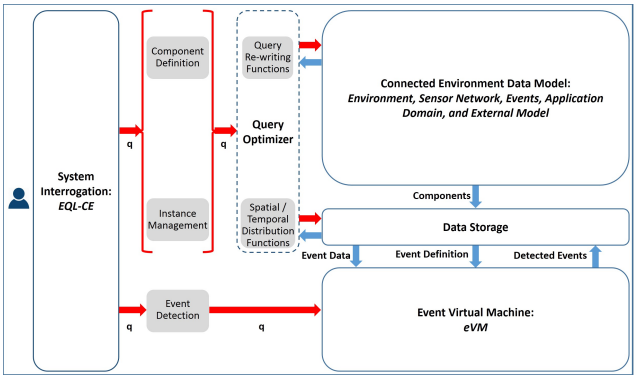


Figure 3: Global Framework Overview

Figure 3 illustrates our event detection framework. It contains three main modules:

- An event query language for connected environments (EQL-CE) and its query optimizer.
- A data model for connected environment representation.
- An event Virtual Machine for event detection (eVM).

4.1.1 Event Query Language. Users interrogate the system using the event query language. It is pivotal since it affects both the data model and the event Virtual Machine (event detector) modules. EQL-CE offers queries that can be used to define the structures of the data model components (i.e., entities that represent the connected environment). In addition, the language allows users to import external data models in the framework. Once the data model is defined, it is saved in the data storage. EQL-CE also manages instances of the previously defined components. It supports operations such as inserting new instances or even modifying, deleting, and retrieving existing ones. Also, the security and privacy of data/users can also be provided by EQL-CE via specific queries. From an event detection standpoint, users can trigger the event Virtual Machine via the query language in order to detect specific events. Finally, the query optimizer allows re-writing queries when needed, and can integrate spatial/temporal distribution functions in the queries (cf. Criterion 3 and 4). Both EQL-CE and its query optimizer will be further detailed in the following subsection.

4.1.2 Data Model. The data model of the connected environment gathers components that describe the environment itself, the sensor network, the events, and the application domain. When considering the environment, one might represent physical, real world, infrastructures such as buildings or offices and all their characteristics. This includes spatial descriptors (e.g., location maps, zones, individual locations, spatial relations), and specific entities that can be found in the environment (e.g., machines, equipment, devices). When considering the sensor network, one might represent sensors, observable properties, scalar/multimedia data, and so on. The targeted events should also be defined and described in the model. This includes event features, types, and patterns. Finally, the application domain is also a part of the model since it affects both the events and the environment. For instance medical events (e.g., high heart rate) differ from environmental events (e.g., temperature overheat in a room). Similarly, the equipment and entities found in a mall are different from the ones found in a hospital.

4.1.3 Event Detector. We proposed the event Virtual Machine (eVM) in a previous work [16]. It is an event detector that needs an event definition and a set of data objects (e.g., sensor observations) in order to detect targeted events. eVM is re-usable in different contexts, extensible, accepts various datatypes, easy to integrate, and requires low human intervention. The event detection process starts by retrieving the targeted event definition form the storage unit. The event definition is analyzed first in order to check its describing features. For instance a fire event is described by the following features: time, location, temperature, smoke, and CO_2. Then, the pre-processor retrieves data objects (e.g., sensor observations) having attributes related to these features (e.g., smoke, temperature, CO_2 observations). Once this is done, we use Formal Concept Analysis (FCA), a conceptual clustering technique, in order to construct a graph from the selected data objects/attributes. Finally, we detect the targeted events by examining the graph nodes and selecting the ones that are compatible with the event definition. Also, eVM is pluggable in the framework and can be replaced by any other event detector that requires data and an event definition in order to detect events. We do not detail the event detection mechanism in this paper since the aim is to focus on the event query language for connected environments.

4.2 The EQL-CE Proposal

We structure our proposal into three layers: (i) the conceptual layer provides an overview of the connected environment's components and their relations in the form of a graph; (ii) the logical layer allows the construction of generic queries written in EBNF (Extended Backus Norm Form) syntax; and (iii) the physical layer parses the EBNF queries into a data model-specific language (e.g., SQL, SPARQL) and executes the parsed queries. A simplified overview of EQL-CE is presented in Figure 4. In the following we detail each layer separately.

4.2.1 ***Conceptual Layer.*** Here, we detail the top layer of EQL-CE. The aim is to provide a clear and easy to exploit conceptual view of the connected environment. Therefore, we use a graph to represent the various elements (i.e., components and properties). The latter are split into the following categories (cf. Figure 5):

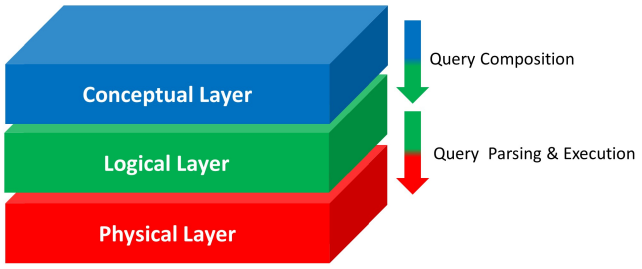


Figure 4: EQL-CE Overview

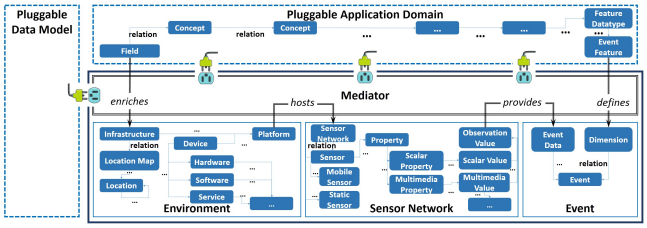


Figure 5: EQL-CE Conceptual Layer

Core Modeling: This part contains the basic elements that always exist in a connected environment. For a clear organization, we group the elements into the following two parts:

- Sensor Network modeling, where we represent (i) *sensor networks*; (ii) various *sensor* types (e.g., static, mobile); (iii) the different types of *properties* (i.e., scalar, multimedia) observed by sensors; and (iv) the *observation values* produced by sensors (i.e., textual values, multimedia objects and their respective metadata).
- Environment modeling, where we represent (i) *platforms* (i.e., infrastructure, devices) that host sensors or sensor networks; (ii) physical *infrastructures*, such as buildings, and their detailed description (i.e., *location maps*, spatial relations); (iii) *devices*, such as mobile phones, and their detailed description (i.e., *hardware*, *software*, provided *services*).

Many other components can still be added to the core part. The full description of the environment and sensor network can be inspired from ontologies such as SOSA/SSN [12] and HSSN (Hybrid Semantic Sensor Network).[1]

Event Modeling: This part contains the representation of events that one might wish to detect in a connected environment. Here, the application domain should also be considered since it affects the definition of specific events. For instance a body overheating (medical) event cannot be defined the same way as a room overheating (environmental) event. Hence, the application domain dictates the type of an event, its describing features, its pattern, and the required data for its detection. Therefore, we do not detail the event modeling, we keep it generic and restrict it to the following components: (i) *event* that defines an event and its type; (ii) *dimensions* to mathematically represent the event features (provided by the Application Domain) in a n-dimensional space; and (iii) *event data* to represent sensor observations that contributed in each event. This allows us to have a generic event definition that applies to various events from different application domains. All context specific details are defined in the application domain and then imported in the event definition via the mediator.

Application Domain Modeling: This part represents the application domain (e.g., medical, energy, military). Since these elements differ from one field to another, this part is pluggable into the conceptual model. It contains basic components/properties denoted *concepts* and *relations* respectively. Instances of the *concept* component can be used to define any domain specific entities, and instances of the *relation* property can be used to interconnect the *concepts* (e.g., Figure 5 shows an *Event Feature* concept that helps define event *dimensions*). This allows the customization of environment descriptions and event definitions based on specific contexts. For instance, one might wish to represent medical equipment and health related constraints when modeling a hospital environment. These elements are not the same when describing a shopping center. Similarly, what describes medical related events is different from normal every day events that happen in a mall. To conclude, this part of the data model complements the event description on one side, and enriches the environment representation on the other.

The Mediator: This part of the conceptual model only contains properties that ensure the interconnection of the previously mentioned parts (i.e., the core, event, and application domain). For instance, a *platform hosts a sensor network*, the *observation values* produced by the sensors *provide event data*, the event *dimensions* are defined by *event features*, and the concept *field enriches* the description of an *infrastructure* based on the application domain. In addition, the mediator can also be used to plug in an external data model and align it with the existing elements.

4.2.2 ***Logical Layer.*** The middle layer of EQL-CE, denoted the logical layer, allows users to compose/design their queries. The process starts by choosing a specific query type. To cover a wider set of functionality (cf. Criterion 1), we provide three main groups of queries:

- The Component Definition Language defines the structure of components. Various query types are included in this group (e.g., CREATE, ALTER, RENAME, DROP).
- The Component Manipulation Language handles component instances. Here we propose the following query types: SELECT, INSERT, UPDATE, and DELETE.
- Component Access Control (e.g., GRANT, REVOKE). These queries manage access rights to component data. We detail access control tasks in a dedicated future work.

The process of composing a query is described in Figure 6. First, the user chooses the query type (e.g., CREATE, INSERT, DELETE). Then, the user starts filling the mandatory statements (e.g., what to CREATE, what to SELECT, from which component). Once this is done, the user can add optional statements for filtering, ordering, calling external functions. Finally, the query is written using an Extended Backus-Naur Form syntax, denoted EBNF [22]. This context-free grammar is used to formally describe programming languages. It extends the Backus-Naur Form (BNF). We use EBNF since it allows the conception of technology independent queries

[1] http://spider.sigappfr.org/research-projects/hybrid-ssn-ontology/

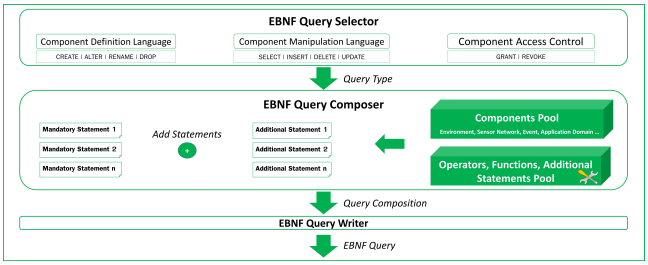


Figure 6: EQL-CE Logical Layer

(i.e., queries that do not depend from any data model specific syntax). This highlights the ability to re-use (cf. Criterion 2) EQL-CE in different setups, since EBNF can later be parsed, in the physical layer, to a specific data model instance, such as SQL or SPARQL, depending on the underlying infrastructure [13, 18, 21]. Any component from the conceptual model (i.e., related to the environment, sensor network, event, and application domain modeling) can be defined, manipulated, and protected using these queries (cf. Criterion 1). Finally, the EBNF query is sent to the physical layer.

4.2.3 ***Physical Layer & Query Optimizer.*** The bottom layer of EQL-CE (cf. Figure 7) saves the received EBNF queries in a dedicated storage unit for future use. Then, it parses the aforementioned queries into a specific syntax depending on the underlying data model (e.g., SQL, SPARQL). Finally, the parsed query is saved and sent to the query run engine where it is executed. If needed, external functions, methods, or even algorithms are called (e.g., string comparison functions, mathematical libraries). All the above de-

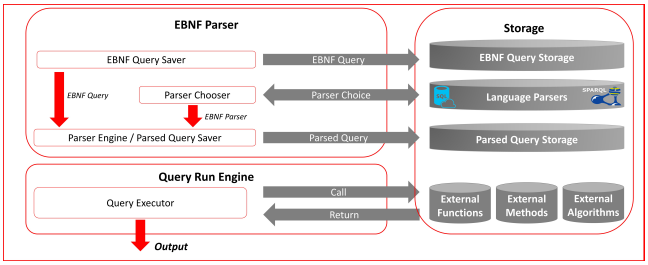


Figure 7: EQL-CE Physical Layer

scribes how EQL-CE can be re-used in various contexts, since it is independent from any technological infrastructure (cf. Challenge 3). Using the EBNF queries, one can define the data model and all its various related components (cf. Challenge 1). In addition, EQL-CE allows users to handle instances of each component for data retrieval, modification, deletion, security/privacy, and event detection by providing a plethora of functionality (cf. Challenge 2). However, when defining specific events, one might need to manage the spatial distribution of sensors over a location (cf. Need 2). For instance, consider k-nearest sensors to a specific location, or all sensors within a range R of a point in space. Also, one might consider mathematical distributions of sensors over a zone (e.g., even distribution). Similarly, one might need to manage the temporal distribution of sensor observations for specific events (cf. Need 3). For example, selecting the k-most recent sensor observations, or all observations that were produced during a specific time lapse. Also, one might need to select observations based on specific sensing rates. To do so, the query optimizer allows the integration of spatial/temporal distribution functions in the queries (cf. Criterion 3 and Challenge 4). Finally, in dynamic connected environments sensors might suffer from breakdowns, mobile sensors could enter/leave the network, or even change locations. This is challenging since event definitions rely on sensors and their provided observations. Hence, some previously defined events might become obsolete over time. Therefore, in some cases, queries need to be re-written or updated in order to handle the dynamicity of the environment, and keep up with its evolution (cf. Criterion 4 and Challenge 5). This is also possible via the query optimizer. In this paper, we do not fully detail the query re-writing and spatial/temporal distribution functions. We leave this to a dedicated future work.

5 ILLUSTRATION EXAMPLE

In this section, we illustrate how EQL-CE works. The aim here, is to demonstrate the component syntax and provide some EBNF query examples. To do so, we consider the smart mall scenario of Section 2. For the sake of brevity, we do not define the entire connected environment (e.g., all locations, sensors in the mall). A fully detailed example (i.e., containing various query types and components) can be found on the following link: http://spider.sigappfr.org/research-projects/eql-ce/smart-mall/.

5.1 Environment Modeling

The mall is an infrastructure having a location map and various locations (e.g., shop 1, food court) that are tied by spatial relations. First, we need to define these components. Then, we INSERT instances. Syntax 1 defines an infrastructure as an entity that has a location map, and a set of embedded platforms (e.g., infrastructures, devices). A location map contains various locations (cf. Syntax 2). Finally, each location can be spatially tied to other locations (cf. Syntax 3).

Syntax 1: Defining the structure of an Infrastructure

```
CREATE INFRASTRUCTURE ( <id> = <string> ,
[ LOCATION MAP <id>  = <string> ,] [ { HOSTED PLATFORM <id>  =  <string> } ] ) ;
```

Syntax 2: Defining the structure of a Location Map

```
CREATE LOCATION MAP ( <id> = <string> , [ { LOCATION <id> =  <string> } ] ) ;
```

Syntax 3: Defining the structure of a Location

```
CREATE LOCATION ( <id> = <string> ,
[ { RELATION TYPE <relation_type> = 'directional'|'distance'|'topological',
   RELATION NAME <relation_name> = 'above'|'below'|'leftOf'|'opposite'|
    'rightOf'|'closeTo'|'farFrom'|'contains'|'covers'|'crosses'|
    'disjoint'|'equals'|'overlaps'|'touches',
    OTHER LOCATION <id> = <string> } ] ) ;
```

In addition, one can rename, drop, or alter component definitions. We give an example for each of these queries in the following:

Syntax 4: Renaming a component

```
RENAME COMPONENT <id> = <string>, <new_id> = <string> ;
```

Syntax 5: Dropping a component

```
DROP COMPONENT <id> = <string> ;
```

Syntax 6: Altering the Location component (add a description field)

```
ALTER LOCATION ( <id> = <string> ,
ADD [ DESCRIPTION <description> = <string> ,] ) ;
```

Once the components' definitions are established, we can start creating instances using INSERT queries. Queries 1, 2, and 3 instantiate an infrastructure, a location map, and location components respectively. We do not cover all locations found in Figure 1 to avoid redundancies.

Query 1: Inserting an Infrastructure instance

```
INSERT INFRASTRUCTURE HAVING ( <id> = 'Mall Infra',
LOCATION MAP <id>  = 'Mall Map' ) ;
```

Query 2: Inserting an Location Map instance

```
INSERT LOCATION MAP HAVING ( <id> = 'Mall Map' ,
LOCATION <id> = 'Shop 1', 'Movie Theatre' ) ;
```

Query 3: Inserting two Location instances

```
INSERT LOCATION HAVING ( <id> = 'Shop 1' ,
RELATION TYPE <relation_type> = 'directional',
RELATION NAME <relation_name> = 'leftOf',
OTHER LOCATION <id> 'Movie Theatre' ) ;

INSERT LOCATION HAVING ( <id> = 'Movie Theatre' ,
RELATION TYPE <relation_type> = 'directional',
RELATION NAME <relation_name> = 'rightOf',
OTHER LOCATION <id> 'Shop 1' ) ;
```

In addition, one can select, update, or delete instances of components. We give an example for each of these queries in the following:

Query 4: Selecting all Locations from the Location Map

```
SELECT LOCATION <id> FROM LOCATION MAP WHERE
LOCATION MAP <id> = 'Mall Map';
```

Query 5: Updating the location relation between Shop 1 and Movie Theatre

```
UPDATE LOCATION CHANGE RELATION NAME <relation_name> = 'leftOf'
INTO RELATION NAME <relation_name> = 'opposite',
WHERE ( LOCATION <id> = 'Shop 1', OTHER LOCATION <id> = 'Movie Theatre');
```

Query 6: Deleting a Location

```
DELETE LOCATION WHERE LOCATION <id> = 'Shop 1';
```

This concludes the definition and manipulation syntax/queries for the environment part. Next, we discuss sensor networks, events, and application domains. Due to space limitations, we focus mainly on the syntax to define the components' structures.

5.2 Sensor Network Modeling

The sensor network hosted in the mall comprises of various static and mobile sensors. They monitor the environment properties and produce observations. Some properties/observations are scalar (e.g., temperature) while others are multimedia (e.g., video surveillance). Therefore, we define here the following components: (i) Scalar Property; (ii) Media Property; (iii) Scalar Value; (iv) Media Value; and (v) Sensor. Syntax 7 details the structure of a scalar property which is mapped to a set of scalar observation values. Similarly, a media property (cf. Syntax 8) is mapped to a set of media values and a specific type (e.g., audio, video, image). Syntax 9 defines any scalar observation value produced by a sensor. Each observation has a timestamp, location, related sensor, a datatype, a value, and a unit. Media observation values are detailed in Syntax 10. Each media value is composed of a data object and a set of metadata/value pairs. Similarly to scalar values, each media value has a timestamp, location, and a related sensor. Finally, a sensor is defined as en entity that has a type (e.g., static, mobile), a current location/coverage area, a set of previous locations/coverage areas/capabilities. Each sensor is capable of sensing specific properties and can be hosted on a particular platform (a device or an infrastructure). Syntax 11 describes the sensor component structure.

Syntax 7: Creating a Scalar Property

```
CREATE SCALAR PROPERTY ( <id> = <string> ,
[ { SCALAR VALUE <id> = <string> } ] ) ;
```

Syntax 8: Creating a Media Property

```
CREATE MEDIA PROPERTY ( <id> = <string> ,
[ MEDIA TYPE <id> = 'audio' | 'image' | 'video'  , ]
[ { MEDIA VALUE <id> = <string> } ] ) ;
```

Syntax 9: Creating a Scalar Value

```
CREATE SCALAR VALUE ( <id> = <string> ,
[ DATATYPE <dt> = 'integer' | 'float' | 'boolean' | 'date' | 'time' |
'date time' | 'character' | 'string' ,  VALUE <val> = <empty> , ]
[ UNIT <id> = <string> , ] [ TIMESTAMP <val> = <empty> , ]
[LOCATION <location_id> = <empty> , ] [ SENSOR <sensor_id> = <empty> ] ) ;
```

Syntax 10: Creating a Media Value

```
CREATE MEDIA VALUE ( <id> = <string> ,
[ DATA OBJECT TYPE <dot> = 'audio segment'|'visual segment' ,
DATA OBJECT <do> = <empty> , ]
[ { METADATA <meta> = 'text annotation descriptor'|'fundamental frequency'|
'harmonic descriptor'|'harmonic spectral centroid'|
'harmonic spectral deviation'|'harmonic spectral spread'|
'harmonic spectral variation'|'log attack time'|'power descriptor'|
'spectral centroid'|'spectrum basis'|'spectrum centroid'|
'spectrum envelop'|'spectrum flatness'|'spectrum projection'|
'spectrum spread'|'temporal centroid'|'waveform'|'camera motion descriptor'|
'motion activity descriptor'|'parametric motion descriptor'|
'trajectory descriptor'|'warping parameters'|'bounding box descriptor'|
'point descriptor'|'media duration descriptor'|'media time point descriptor'|
'color layout descriptor'|'color structure descriptor'|
'contour shape descriptor'|'dominant color descriptor'|
'edge histogram descriptor'|'face recognition descriptor'|
'scalable color descriptor' ,  VALUE <val> = <empty> } ,]
[ TIMESTAMP <val> = <empty> , ] [LOCATION <location_id> = <empty> , ]
[ SENSOR <sensor_id> = <empty> ] ) ;
```

Syntax 11: Creating a Sensor

```
CREATE SENSOR ( <id> = <string> ,
( [ HAVING
[ SENSOR TYPE <sensor_type> = 'static' | 'mobile' , ]
[ CURRENT LOCATION <id> = <string> , ]
[ { PREVIOUS LOCATION <id> = <string> , TIME INTERVAL <ti> = <empty> } , ]
[ CURRENT COVERAGE AREA <id> = <string> ,  ]
[ { PREVIOUS COVERAGE AREA <id> = <string> , TIME INTERVAL <ti> = <empty> } , ]
[ { CAPABILITY <id> = <string> , VALUE <val> = <string> } ] ] , )
( [ SENSING { SCALAR PROPERTY <id> = <string> |
              MEDIA PROPERTY <id> = <string> } ] , )
( [ HOSTED ON PLATFORM <id> = <string> ] ) ) ;
```

5.3 Event Modeling

Here we detail the event modeling in EQL-CE. We define the event as a n-dimensional space where each dimension mathematically represents an event describing feature. The latter are provided by the application domain (cf. Figure 5). Moreover, event data is the set of sensor observations that help detect the event (i.e., event data belong to the event's n-dimensional space). Therefore, an event has a set of dimensions and event data. In addition, an event also has a set of sensors that provide the required observations for the detection. Finally, we added a type parameter to the event definition to distinguish elementary or atomic events (i.e., that require one observation from one sensor) from complex events (i.e., that require various observations from one sensor), and composite ones (i.e., that require various observations from different sensors). The following syntax defines event modeling components.

Syntax 12: Creating an Event Structure

```
CREATE EVENT ( <id> = <string> ,
[ EVENT TYPE <event_type> = 'elementary' | 'complex' | 'composite' , ]
[ { SENSOR <sensor_id> = <string> } , ]
[ { DIMENSION <dimension> = <string> } , ]
[ { EVENT DATA <data_object> = SCALAR OBSERVATION <so> |
                               MEDIA OBSERVATION <mo> } ] ) ;
```

The following query defines a particular event instance, denoted 'Overheat in Shop 1', where the three main dimensions are time, location, and temperature. This event relies on scalar temperature observations that surpass the value 30. Once the event instance is defined, any external event detection mechanism (e.g., eVM cf. Figure 3 can use this definition to detect occurrences of this event.

Query 7: Creating an Event Instance

```
INSERT EVENT HAVING ( <id> = 'Overheat in Shop 1' ,
EVENT TYPE <event_type> = 'elementary',
{ SENSOR <sensor_id> = ANY },
{ DIMENSION <dimension> = 'Time', 'Location', 'Temperature' },
{ EVENT DATA <data_object> = SCALAR OBSERVATION <so> } ),
WHERE ( <so>.<id> = 'Temperature',
        <so>.<location_id> = 'Shop 1', <so>.<val> > 30 ) ) ;
```

To keep up with the environment changes (cf. Criterion 4), one could need to re-write obsolete event definitions. Query re-writing is provided automatically by the query optimizer (cf. Figure 3). However, users can request an update at any time. This is illustrated in the following query where we update the event definition provided in Query 7 by only considering observations from Sensor 1.

Query 8: UPDATING an Event Instance

```
UPDATING EVENT CHANGE (
SENSOR <sensor_id> = 'Sensor 1',
WHERE (EVENT <id> = 'Overheat in Shop 1') ) ;
```

5.4 Application Domain Modeling

As previously mentioned in the conceptual layer, application domains have different components, inter-component relations, and targeted events. Therefore, we provide here a generic definition of an application domain related components and relations (denoted Concept, Relation respectively cf. Syntax 13). We also provide a definition for event describing features (cf. Syntax 14) and their datatypes (cf. Syntax 15) that can be instantiated in different domains.

Syntax 13: Creating a Concept/Relation

```
CREATE CONCEPT ( <id> = <string> , [ { ATTRIBUTE <id> } ] ) ;
ATTRIBUTE <id> = CONCEPT <id> | VARIABLE ( <label> = <string> ,
DATATYPE <datatype> = 'integer'|'float'|'boolean'|'date'|'time'|
'date time'|'character'|'string', VALUE <val> = <empty> ) ;

CREATE RELATION ( <id> = <string> ,
[ { CONCEPT <source_id> = <string> } , ]
[ { CONCEPT <target_id> = <string> } ] ) ;
```

Every event feature has an identifier, a set of granularities (e.g., second, minute, hour for time), a function that converts a granularity to another (e.g., 1 minute = 60 seconds), a boolean field indicating if intervals can be created from this feature's values, and a datatype.

Syntax 14: Creating an Event Feature

```
CREATE EVENT FEATURE ( <id> = <string> ,
[ GRANULARITY SET { VALUE <val> = <string> } , ]
[ GRANULARITY FUNCTION <id> = <string> , ]
[ INTERVAL <boolean> = '0' | '1' , ]
[ EVENT FEATURE DATATYPE <event_feature_datatype_id> = <string> ] ) ;
```

An event feature datatype has an identifier, a primitive datatype , a range of allowed values (i.e., lower bound min, upper bound max), and a function that measures the distance between values having the same event feature datatype. These details help translate event describing features (application domain) into dimensions of the event's n-dimensional space (event modeling) using the mediator (cf. Figure 5).

Syntax 15: Creating a Event Feature Datatype

```
CREATE EVENT FEATURE DATATYPE ( <id> = <string> ,
[ DATATYPE <datatype> = 'integer'|'float'|'boolean'|'date'|'time'|
'date time'|'character'|'string', ]
[ RANGE ( MIN <min> = VALUE <val> , MAX <max> = VALUE <val> ) , ]
[ DISTANCE <function_id> = <string> ] ) ;
```

6 CONCLUSION & FUTURE WORK

Many challenges emerge when proposing a EQL adapted to connected environments. In this paper, we addressed the issues of re-usability, and covering various components/functionality. To do so, we proposed EQL-CE: a three layered event query language for connected environments. We detailed its conceptual, logical, and physical layers. EQL-CE users compose EBNF queries, that can be

later parsed into SQL, SPARQL, or other languages (re-usability). Our proposal covers various connected environment components (environments, sensor networks, events, and application domains) and functionality (definition, manipulation, access control, event detection). We also proposed a query optimizer that allows query re-writing and the integration of spatial/temporal distribution functions. As future work, we would like to detail the security/privacy related queries and distribution/query re-writing functions. Also, we are currently developing an online simulator to allow users to run tests on a connected environment (e.g., the smart mall). Finally, we want to address additional challenges such as integrating batch queries and continuously processing data streams.

REFERENCES

[1] Yuvraj Agarwal, Bharathan Balaji, Rajesh Gupta, Jacob Lyles, Michael Wei, and Thomas Weng. 2010. Occupancy-driven energy management for smart building automation. In *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building*. ACM, 1–6.

[2] Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. 2011. EP-SPARQL: a unified language for event processing and stream reasoning. In *Proceedings of the 20th international conference on World wide web*. ACM, 635–644.

[3] Darko Anicic, Paul Fodor, Sebastian Rudolph, Roland Stühmer, Nenad Stojanovic, and Rudi Studer. 2011. Etalis: Rule-based reasoning in event processing. In *Reasoning in event-based distributed systems*. Springer, 99–124.

[4] Arvind Arasu, Shivnath Babu, and Jennifer Widom. 2006. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal* 15, 2 (2006), 121–142.

[5] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. 2009. C-SPARQL: SPARQL for continuous querying. In *Te 18th international conference on World wide web-WWW'09*. 1061–1062.

[6] Roger S Barga and Hillary Caituiro-Monge. 2006. Event correlation and pattern detection in CEDR. In *International Conference on Extending Database Technology*. Springer, 919–930.

[7] François Bry and Michael Eckert. 2007. Rule-based composite event queries: the language XChange EQ and its semantics. In *International Conference on Web Reasoning and Rule Systems*. Springer, 16–30.

[8] AH Buckman, Martin Mayfield, and Stephen BM Beck. 2014. What is a smart building? *Smart and Sustainable Built Environment* 3, 2 (2014), 92–109.

[9] Sharma Chakravarthy and Deepak Mishra. 1994. Snoop: An expressive event specification language for active databases. *Data & Knowledge Engineering* 14, 1 (1994), 1–26.

[10] EsperTech. [n. d.]. EsperTech. Chapter 5. EPL reference:Clauses. http://esper.espertech.com/release-5.3.0/esper-reference/html_single/index.html#epl_clauses. Accessed: 2019-02-07.

[11] Daniel Gyllstrom, Eugene Wu, Hee-Jin Chae, Yanlei Diao, Patrick Stahlberg, and Gordon Anderson. 2006. SASE: Complex event processing over streams. *arXiv preprint cs/0612128* (2006).

[12] Armin Haller, Krzysztof Janowicz, Simon JD Cox, Maxime Lefrançois, Kerry Taylor, Danh Le Phuoc, Joshua Lieberman, Raúl García-Castro, Rob Atkinson, and Claus Stadler. 2018. The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation. *Semantic Web* Preprint (2018), 1–24.

[13] Konstantinos Kemalis and Theodores Tzouramanis. 2008. SQL-IDS: a specification-based approach for SQL-injection detection. In *Proceedings of the 2008 ACM symposium on Applied computing*. ACM, 2153–2158.

[14] Timilehin Labeodan, Christel De Bakker, Alexander Rosemann, and Wim Zeiler. 2016. On the application of wireless sensors and actuators network in existing buildings for occupancy detection and occupancy-driven lighting control. *Energy and Buildings* 127 (2016), 75–83.

[15] Jay Lee, Behrad Bagheri, and Hung-An Kao. 2015. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters* 3 (2015), 18–23.

[16] Elio Mansour, Richard Chbeir, and Philippe Arnould. 2018. eVM: An Event Virtual Machine Framework. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXIX*. Springer, 130–168.

[17] Matthew Perry, Prateek Jain, and Amit P Sheth. 2011. Sparql-st: Extending sparql to support spatiotemporal queries. In *Geospatial semantics and the semantic web*. Springer, 61–86.

[18] Oumy Seye, Catherine Faron-Zucker, Olivier Corby, and Corentin Follenfant. 2012. Bridging the Gap between RIF and SPARQL: Implementation of a RIF Dialect with a SPARQL Rule Engine. *AImWD 2012* (2012), 19.

[19] Winda Astuti Wahyudi and M Syazilawati. 2007. Intelligent voice-based door access control system using adaptive-network-based fuzzy inference systems (ANFIS) for building security. *Journal of Computer Science* 3, 5 (2007), 274–280.

[20] James Welch, Farzin Guilak, and Steven D Baker. 2004. A wireless ECG smart sensor for broad application in life threatening event detection. In *Engineering in Medicine and Biology Society, 2004. IEMBS'04. 26th Annual International Conference of the IEEE*, Vol. 2. IEEE, 3447–3449.

[21] Niklaus Wirth. 1977. What can we do about the unnecessary diversity of notation for syntactic definitions? *Commun. ACM* 20, 11 (1977), 822–823.

[22] Niklaus Wirth, Niklaus Wirth, Niklaus Wirth, Suisse Informaticien, and Niklaus Wirth. 1996. *Compiler construction*. Vol. 1. Citeseer.

[23] Johnny KW Wong, Heng Li, and SW Wang. 2005. Intelligent building research: a review. *Automation in construction* 14, 1 (2005), 143–159.

[24] Liyang Yu, Neng Wang, and Xiaoqiao Meng. 2005. Real-time forest fire detection with wireless sensor networks. In *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on*, Vol. 2. IEEE, 1214–1217.

[25] S Zampolli, I Elmi, F Ahmed, M Passini, GC Cardinali, S Nicoletti, and L Dori. 2004. An electronic nose based on solid state sensor arrays for low-cost indoor air quality monitoring applications. *Sensors and Actuators B: Chemical* 101, 1-2 (2004), 39–46.