

Semantic Discovery and Integration of Urban Data Streams ^{*}

Feng Gao, Muhammad Intizar Ali and Alessandra Mileo

Insight Centre for Data Analytics,
National University of Ireland, Galway, Ireland
{feng.gao, ali.intizar, alessandra.mileo}@insight-centre.org

Abstract. With the growing popularity of Internet of Things (IoT) technologies and sensors deployment, more and more cities are leaning towards the initiative of smart cities. Smart city applications are mostly developed with aims to solve domain-specific problems. Hence, lacking the ability to automatically discover and integrate heterogeneous sensor data streams on the fly. To provide a domain-independent platform and take full benefits from semantic technologies, in this paper we present an *Automated Complex Event Implementation System (ACEIS)*, which serves as a middleware between sensor data streams and smart city applications. ACEIS discovers and integrates IoT streams in urban infrastructures for users' requirements expressed as complex event requests, based on semantic IoT stream descriptions. It also processes complex event patterns on the fly using semantic data streams.

1 Introduction

An increasing number of cities have started to embrace the idea of smart cities and are in process of building smart city infrastructure for its citizens. Such infrastructures, including the deployment of sensors, provision of open data platforms and smart city applications, can improve the day to day life for the citizens. A typical example of smart city applications is the provision of real-time tracking and timetable information for the public transport within the city¹. City of Aarhus provides an open data platform called ODAA², which contains city related information generated by various sensors deployed within the city, e.g., traffic congestion level, air quality and trash-bin level etc. ODAA also encourages usage of their open data platform for building smart city applications. In the foreseeable future, more and more urban data will be made available. The enormous amount of the data produced by sensors in our day to day life need to be harnessed to help smart city applications taking smart decisions on the fly.

^{*} This research has been partially supported by Science Foundation Ireland (SFI) under grant No. SFI/12/RC/2289 and EU FP7 CityPulse Project under grant No.603095. <http://www.ict-citypulse.eu>

¹ Live bus arrivals in London: <http://countdown.tfl.gov.uk/#/>

² Open Data Aarhus: <http://odaa.dk>

However, the uptake of smart city applications is hindered by various issues, such as difficulty of discovering the capabilities of the available infrastructure and once discovered, integrating heterogeneous data sources and extracting up-to-date information in real-time. The smart city data needs to be integrated from various domains in a federated fashion. Integrated information should be further processed, aggregated and higher-level abstractions should be created from the data to make it suitable for complex event processing in real-time. Existing semantic service discovery and composition approaches (e.g., WSMO³, OWL-S⁴) are based on *Input, Output, Precondition and Effect*. They are not suitable for describing complex event processing services with event patterns. Moreover, IoT streams are inherently dynamic and resource constrained. Providing support for quality-aware distributed event systems consuming IoT streams is still a challenge [7, 16].

In this paper, we present ACEIS, which is an automated discovery and integration system for urban data streams. We design a semantic information model to represent complex event services and utilize this information model for the discovery and integration of sensor data streams. ACEIS assumes that all available sensor data streams are annotated using Semantic Sensor Network (SSN)⁵ and stored in a repository. Various Quality of Service (QoS) and Quality of Information (QoI) metrics are also annotated for each sensor data stream. ACEIS receives an event service request described using our complex event service information model and automatically discovers and composes the most suitable data streams for the particular event request. ACEIS then transforms the event service composition into a stream query to be deployed and executed on a stream engine to evaluate the complex event pattern specified in the event service request. The contributions of this paper can be summarised as below:

- * We present an Automated Complex Event Implementation System serving as a middleware between Smart City applications and sensor data streams.
- * We introduce an information model (Complex Event Service Ontology) and demonstrate its usage in describing semantic event services and service requests.
- * We elaborate the mechanisms for QoS-aware discovery and integration of semantic event services.
- * We implement an automatic query transformation system to formulate continuous queries over semantic sensor data streams.

Structure of the Paper: In Section 2, we lay the foundations of our study by identifying various types of sensor data streams and challenges faced by smart city applications while using these sensor data streams. We presented a conceptual architecture of our proposed system (ACEIS) in Section 3. Detailed description of the sensor data streams discovery and integration is provided in Section 4. Section 5 discusses our automated query transformation algorithm. We

³ Web Service Modeling Ontology: <http://www.wsmo.org/>

⁴ OWL-S ontology: <http://www.w3.org/Submission/OWL-S/>

⁵ SSN ontology: <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn>

positioned our work by comparing it with state of the art in Section 6 before concluding in Section 7.

2 Smart City Applications

In this section, firstly, we discuss different types of sensor data streams which can be potentially utilized by the smart city applications and later we discuss the requirements and challenges faced by these smart city applications.

2.1 Sensor Data Streams

Sensors are nowadays part of our every day life. IoT technologies not only provide an infrastructure for the sensor deployment but also provide a mechanism for better communication among these sensors. Data being produced by these sensors is enormous and there is a strong need to tame these data streams and build applications to take smart decision by performing analysis of these data streams in real-time. Data streams produced by various sensors can be categorised into three different categories:

Physical Sensors: Various sensors are being deployed by city administration with an aim to closely observe and monitor the city infrastructure. Traffic congestion, air quality, temperature, water pressure and trash bin level sensors are few examples of the sensors deployed within most of the modern cities. Additionally, various sensors are being deployed in buildings (e.g. airports, train stations) to detect critical events happened therein.

Mobile and Wearable Sensors: Contrary to the physical sensors deployed by city administrations and organizations, sensors attached to mobile devices provide additional information about the context of their carrier. Nowadays, a modern smart phone, owned and carried by majority of the citizens in the smart cities is equipped with 10 to 15 sensors on average, including location, temperature, light and proximity sensors. Modern cars also contain plenty of sensors to continuously monitor the performance as well as to provide assistance to the users. Many wearable sensors are gaining popularity and many people are adopting to the use of wearable sensors particularly in the health care domain.

Virtual Sensors (Social Media Data Streams): Virtual sensors are usually deployed by integrating multiple physical sensors and provide a cost effective alternative. Social media data streams can also be considered as virtual sensors. Social media data streams are a major source of information in smart city infrastructure and mostly provide latest information about the city events, e.g. Twitter feeds can provide latest information about the city events like accident and traffic jams etc. Although, trust, reliability and provenance are major concerns over the information arising from social media streams, but various social streams analysis methods have been already developed to overcome these concerns.

2.2 Requirements and Challenges

Smart city applications face many challenges because of highly distributed and dynamic nature of the IoT infrastructures deployed in the smart cities. Below we discuss few of the requirements and challenges faced by smart city applications.

Federation of heterogeneous data streams: Data Federation combines heterogeneous sets of data to provide a unified view. In the context of smart city data, data federation is a key challenge due to the dynamicity and heterogeneity of various IoT streams. Querying and accessing the data in many cases will require real-time (or near-real-time) discovery and access to the streams (and their data) and the ability to integrate different kinds of heterogeneous streaming data from various sources. Smart city frameworks should provide mechanisms to (i) seamlessly integrate real world data streams, (ii) automated search, discovery and federation of data streams, and (iii) adaptive techniques to handle fail-overs at run-time.

Large scale IoT processing and data analytics: Smart city applications not only require to efficiently process large scale IoT streams but also need efficient methods to perform data analytics in dynamic environment by aggregating, summarizing and abstracting sensor data on demand. Current analytic frameworks have to be evaluated for applicability in the smart city environment and the impact on privacy has to be taken into account.

Real-time IoT information extraction, event detection and stream reasoning: Smart city applications should be able to process event streams in a real time, extract relevant information and identify values that do not follow the general trends. Beyond the identification of relevant events, extraction of high level knowledge from heterogeneous, multi-modal data streams is an important component of IoT. Existing stream reasoning techniques use background knowledge and streaming queries to reason over data streams. Current techniques of stream reasoning do not cater to the needs of IoT due to the lack of proper treatment of uncertainty (e.g. possible reasons of traffic jam vs. most probable reason of traffic jam) in the IoT environment.

Reliable information processing, QoI, testing and monitoring: Data quality issues and provenance play an important role in smart city scenarios. Smart city frameworks should provide methods and techniques (i) to evaluate accuracy, trustworthiness, and provenance of IoT streams, (ii) to resolve conflicts in case of contradictory information, and (iii) continuous monitoring and testing to dynamically update QoI and trustworthiness.

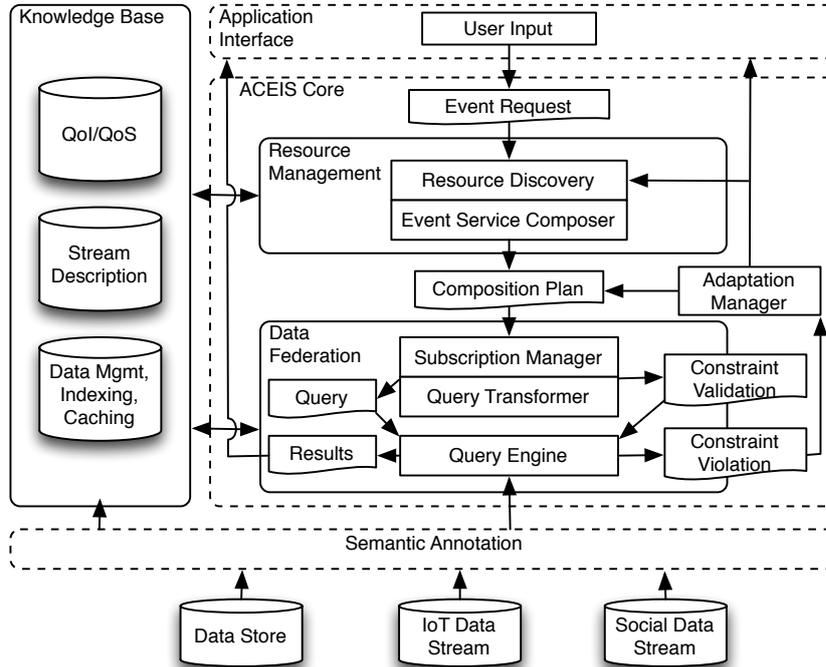


Fig. 1. ACEIS architecture overview

3 Overview of ACEIS Architecture

To address the *data stream federation* and (partially) *reliable information processing* challenges identified in Section 2.2, various solutions are developed and integrated into ACEIS. We will discuss briefly the functionalities of the components in ACEIS as well as their interactions. Figure 1 illustrates the architecture view of ACEIS. The architecture consists of three main components: *Application Interface*, *Semantic Annotation* and *ACEIS Core* component.

3.1 Application Interface

The application interface interacts with end users as well as ACEIS core modules. It allows users to provide inputs required by the application and presents the results to the user in an intuitive way. It also augments the users' queries, requirements and preferences with some additional, implicit constraints and preferences determined by the application domain or user profile. For example, in a travel navigation scenario, a user may specify only the start and target location on the map, with a constraint on the travel time t , because she needs to get there on time. The application may add some additional constraints on the

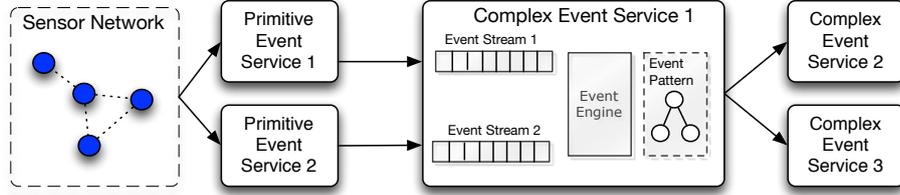


Fig. 2. Example of event service networks (originally published in [5])

IoT data streams used to calculate the travel time, such as the frequency of the data streams should be more than $1/t$, otherwise the user may not receive any updates on the traffic condition during her trip and the detour suggestions for traffic jams will never happen.

These augmented user inputs are transformed into a semantically annotated complex event service request (event request for short). The event request is consumed by ACEIS core components to discover and integrate urban streams w.r.t. the functional and non-functional constraints specified within the event request.

3.2 Semantic Annotation

The semantic annotation component receives IoT/data streams (e.g., ODAA realtime traffic sensors data⁶) as well as static data stores (e.g., ODAA traffic sensors metadata⁷) as inputs. It annotates syntactical information with semantic terms defined in ontologies. The outputs of semantic annotation will be semantic IoT/data streams and static semantic data stores.

With semantic annotations of both static resource and dynamic data, ACEIS gains additional data interoperability both at design time for event service discovery/composition and at runtime for semantic event detection.

3.3 ACEIS Core

The ACEIS core module serves as a middleware between low level IoT data streams and upper level Smart City applications. ACEIS core is capable of discovering, composing, consuming and publishing complex event processing capabilities as reusable services. We call these services (primitive or complex) event services. An example of event service network is shown in Figure 2. ACEIS core consists of two major components: resource management and data federation. In the following, we introduce their functionalities and interactions.

⁶ Realtime Traffic Data in Aarhus: <http://ckan.projects.cavi.dk/dataset/bliptrack-alpha/resource/d7e6c54f-dc2a-4fae-9f2a-b036c804837d>

⁷ Traffic Sensor Metadata: <http://ckan.projects.cavi.dk/dataset/bliptrack-alpha/resource/e132d528-a8a2-4e49-b828-f8f0bb687716>

Resource Management The resource management component is responsible for discovering and composing event services based on static service descriptions. It receives event requests generated by the application interface containing users' functional/non-functional requirements and preferences, and creates composition plans for event requests, specifying which event services are needed to address the requirements in event requests and how they should be composed.

Resource management component contains two sub-components: resource discovery component and event service composer. The resource discovery component uses conventional semantic service discovery technique to retrieve IoT services delivering primitive events. It deals with the primitive event requests specified within event requests. The event service composer creates service composition plans to detect the complex events specified by event requests based on event patterns. We refer readers to [5] for further details of the composition algorithm used by the event service composer.

Data Federation The data federation component is responsible for implementing the composition plan over event service networks and process complex event logics using heterogeneous data sources. The composition plan is firstly used by the subscription manager which will make subscriptions to the event services involved in composition plan. Later, the query transformer transforms the semantically annotated composition plan into a set of stream reasoning queries to be executed on a stream query engine.

The query transformer produces two kinds of stream queries: regular event queries that detect the complex events specified by event requests and constraint validation queries that monitor the constraints specified in event requests. Thus the query engine produces two kinds of results: (i) *event query results* are forwarded to the application interface and (ii) *constraint violations* are detected by constraint validation queries and sent to the adaptation manager. Adaptation manager decides whether an automatic adaptation is possible. If so, it creates and deploys a new composition plan that conforms with the constraints to replace the existing one. Otherwise, dispatches a notification to there application interface.

4 Semantic Sensors Stream Discovery & Integration

In this section, the ontology used for describing event services and event requests are presented, the discovery and integration mechanism for the sensor data streams are discussed.

4.1 Complex Event Service Ontology

A Complex Event Service (CES) ontology is developed to describe event services and requests. CES ontology is an extension of OWL-S. OWL-S is a standardized ontology to describe, discover and compose semantic web services. Figure 3 illustrates the overview of CES ontology. An event service is described with a

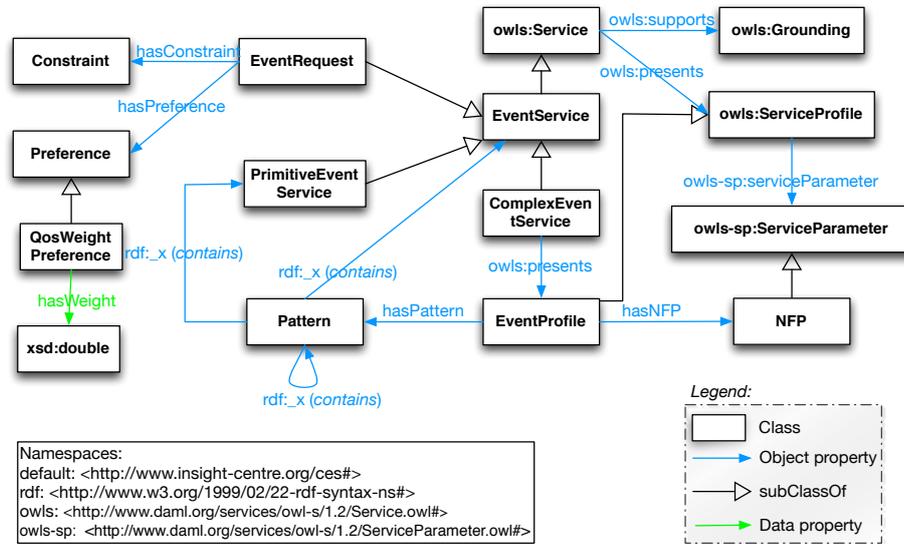


Fig. 3. CES ontology overview

Grounding and an *EventProfile*. The concept of *Grounding* in OWL-S informs an event consumer, how to access the event service by providing information on service protocol and message formats etc. An *EventProfile* is comparable to the *ServiceProfile* in OWL-S, which describes the events transmitted by the service.

An *Event Profile* describes a type of event with a *Pattern* and *Non-Functional Properties* (NFP). A *Pattern* describes the correlations between a set of member events involved in the pattern. An event pattern may have other patterns or (primitive) event services as sub-components, making it a tree structure. An event profile without a *Pattern* describes a primitive event service, otherwise it describes a complex event service. *NFP* refers to the QoI and/or QoS metrics, e.g., precision, reliability, cost and etc, which are modelled as sub-classes of *ServiceParameter* in OWL-S.

An *EventRequest* is specified as an incomplete *EventService* description, without specific bindings to the set of federated event services used by the requested complex event. *Constraints* can be specified by users to declare their requirements on the event pattern and NFPs in *EventRequests*. *Preferences* can be used to specify a weight between 0 to 1 over different quality metrics representing users' preferences on QoS metrics: higher weight indicate the user cares more on the particular QoS metric.

4.2 Sensors Streams Discovery

The task of sensor stream discovery is to find candidate sensor services based on sensor service descriptions and request specifications. A sensor stream is an atomic unit in IoT stream discovery and integration. It is described both as a

PrimitiveEventService in CES ontology, as well as a *Sensor* device in SSN ontology. The CES ontology is mainly used to describe the non-functional aspects of sensor service requests/descriptions, including sensor event types, quality parameters and sensor service groundings. SSN ontology is used to describe the functional aspects, including *ObservedProperties* and *FeatureOfInterest*.

Listing 1. Traffic sensor service description

```

:sampleTrafficSensor a ssn:Sensor, ces:PrimitiveEventService ;
  owls:presents :sampleProfile ;
  owls:supports :sampleGrounding ;
  ssn:observes [ a ces:AverageSpeed ;
                 ssn:isPropertyFor :FoI_1],
               [ a ces:VehicleCount ;
                 ssn:isPropertyFor :FoI_2],
               [ a ces:EstimatedTime ;
                 ssn:isPropertyFor :FoI_3].
:sampleProfile a ces:EventProfile ;
  owls:serviceCategory [ a ces:TrafficReportService ;
  owls:serviceCategoryName "traffic_report"^^xsd:string].

```

Listing 2. Traffic sensor service request

```

:sampleRequest a ssn:Sensor, ces:EventRequest ;
  owls:presents :requestProfile ;
  ssn:observes [ a ces:EstimatedTime ;
                 ssn:isPropertyFor :FoI_3].
:requestProfile a ces:EventProfile ;
  owls:serviceCategory [ a ces:TrafficReportService ;
  owls:serviceCategoryName "traffic_report"^^xsd:string].

```

A sensor service description is denoted as $s_{desc} = (t_d, g, q_d, P_d, FoI_d, f_d)$, where t is the sensor event type, g is the service grounding, q_d is a QoS vector describing the QoS values, P_d is the set of *ObservedProperties*, FoI_d is the set of *FeatureOfInterests* and $f_d : P_d \rightarrow FoI_d$ is a function correlating observed properties with their feature-of-interests. Similarly, a sensor service request is denoted $s_r = (t_r, q_r, P_r, FoI_r, f_r, pref, C)$. Compared to s_d , s_r do not specify service groundings, q_r represents the constraints over QoS metrics, $pref$ represents the QoS weight vector specifying users' preferences on QoS metrics and C is a set of functional constraints on the values of P_r . s_d is considered a match for s_r iff all of the following three conditions are true:

- t_r subsumes t_d ,
- q_d satisfies q_r and
- $\forall p_1 \in P_r, \exists p_2 \in P_d \implies T(p_1)$ subsumes $p_2 \wedge f_r(p_1) = f_d(p_2)$, where $T(p)$ gives the most specific type of p in a property taxonomy.

Listing 1 shows a snippet of a traffic sensor description in turtle syntax. The traffic sensor monitors the estimated travel time, vehicle count and average vehicle speed on a road segment. Listing 2 shows a snippet of an sensor service request

matched by the traffic sensor service. When the discovery component finds all service candidates suitable for the request, a Simple-Additive-Weighting algorithm [4] is used to rank the service candidates based on q_d , q_r and $pref$.

4.3 Sensors Streams Integration

Sensor stream discovery deals only with primitive event service discovery. To discover and integrate (composite) sensor streams for complex event service requests, the event patterns specified in the complex event service requests/descriptions need to be considered.

Listing 3. Complex event service request

```

:SampleEventRequest a ces:EventRequest;
    owls:presents :SampleEventProfile.

:SampleEventProfile rdf:type owls:EventProfile;
    ces:hasPattern [ rdf:type ces:And, rdf:Bag;
        rdf:_1 :locationRequest;
        rdf:_2 :seg1CongestionRequest;
        rdf:_2 :seg2CongestionRequest;
        rdf:_4 :seg3CongestionRequest;
        ces:hasWindow "5"^^xsd:integer];
    ces:hasConstraint [ rdf:type ces:NFPConstraint;
        ces:onProperty ces:Availability;
        ces:hasExpression
        [ emvo:greaterThan "0.9"^^xsd:double]],
        [ rdf:type ces:NFPConstraint;
        ces:onProperty ces:Accuracy;
        ces:hasExpression
        [ emvo:greaterThan "0.9"^^xsd:double]].

```

In the context of integrated sensor stream discovery and composition, the definition of sensor stream description is extended to denote composite sensor stream descriptions $S_d = (ep_d, Q_d, G)$, where ep_d consists of a set of sensor stream descriptions s_d and/or a set of composite sensor stream descriptions S'_d , and a set of event operators including *Sequence*, *Repetition*, *And*, *Or*, *Selection*, *Filter* and *Window*, q_d is the aggregated QoS metrics for S_d and G is the grounding for the composite sensor stream. Similarly, a complex event service request is denoted as $S_r = (ep_r, Q_r, pref)$, where ep_r is a *canonical* event pattern consisting of a set of primitive sensor service requests s_r and a set of event operators, Q_r describes the QoS constraints for the requested complex event service and $pref$ specifies the weights on QoS metrics.

An S_d is a match for S_r iff ep_d is *semantically equivalent* to ep_r and Q_d satisfies Q_r . When no matches are found during the discovery process for S_r , it is necessary to compose S_r with a set of S_d and/or s_d which are *reusable* to S_r . Informally, these (composite) sensor streams describe a part of the semantics of ep_r and can be reused to create a composition plan, which contains an event

pattern with concrete service bindings. The composition plan can be used as a part of the event service description for the composed event service. The discovery or composition results can be ranked w.r.t the QoS metrics and preferences in the same way as sensor stream discovery. We refer readers to [5, 4] for detailed definitions of concepts related to event patterns as well as algorithms to perform an efficient pattern-based and QoS-aware event service discovery and composition. Listing 3 shows a snippet of a sample complex event service request with an event pattern and some NFP constraints.

5 Query Transformation

To implement a composition plan, the subscription manager needs to make subscriptions to the relevant event sources using the service bindings provided in the composition plan. Then, the query transformer creates (regular and constraint validation) stream reasoning queries and registers the queries at the stream engine. In this section, the algorithms for transforming regular queries are discussed.

In the current ACEIS implementation, CQELS[9] is used as the semantic stream reasoning engine. We consume the semantically annotated sensor data streams using SSN ontology. A sample traffic sensor reading annotated as *Observation* in SSN is shown in Listing 4.

Listing 4. Traffic sensor stream data

```

:Observation_1 a ssn:Observation;
                ssn:observedBy :sampleTrafficSensor
                ssn:observedProperty [ a ces:EstimatedTime];
                ssn:featureOfInterest :FoI_1;
                ssn:observationResult :observationResult_1.
:observationResult_1 ssn:hasValue
                    [ ssn:hasQuantityValue "'25'"^^xsd:integer;
                      muo:unitOfMeasurement muo:second].

```

5.1 Semantics Alignment

To ensure the query transformation creates queries that detect the right event patterns, it is required to map the semantics of event operators to query operators. Each event service description s_d or S_d in event patterns should map to a CQELS *StreamGraphPattern* (SGP)⁸ for the *ssn:Observations* transmitted in the event stream. A *Sequence* operator requires its sub-events to occur in a temporal order. Currently CQELS (version 1.0.0) do not provide functions to access the timestamps of the stream triples, therefore *Sequence* is not supported.

⁸ SGP is an extension of GraphPatternNotTriples in SPARQL 1.1 grammar, CQELS language grammar and examples available at: https://code.google.com/p/cqels/wiki/CQELS_language

Repetition is a generalization of sequence, it indicates a sequence pattern should be repeated several times, therefore it is also not supported. An *And* operator indicates all its sub-events should occur, it can be mapped to the *Join* operator. An *Or* operator indicates at least one of its sub-events should occur, it can be mapped to *LeftOuterJoin* operator in CQELS (OPTIONAL keyword) with *bound* filters. *Selection* is mapped to *Projection* in CQELS to select the message payloads for complex events. *Filter* and *Window* operators in event patterns can be mapped to *Filter* and *Window* operators in CQELS, respectively. Table 1 summarizes the semantics alignment between event operators and CQELS operators.

Table 1. Semantics Alignment

Event Pattern	s_d	Sequence	Repetition	And	Or	Selection	Filter	Window
CQELS Operator	SGP	-	-	Join	Optional	Projection	Filter	Window

5.2 Transformation Algorithm

Previously (see Section 4.1), we briefly described how event patterns are specified in CES ontology. An event pattern can be recursively defined with sub event patterns and event service descriptions, thus formulating an event pattern tree. In this section we elaborate algorithms for parsing event pattern trees and creating CQELS queries. In an event pattern tree, the nodes can be event operators in four types: *Sequence*, *Repetition*, *And* and *Or*, or other event service descriptions; the edges represent the provenance relation in the complex event detection: the parent node is detected based on the detection of the child nodes. Using a top-down traversal of the event pattern tree and querying the semantics alignment table for each event operator encountered during the traversal, the event pattern in the composition plan is transformed into a CQELS query following the divide-and-conquer style. Algorithm 1 shows the pseudo code of the main parts of query transformation algorithm.

Lines 1 to 6 in Algorithm 1, construct the CQELS query with three parts: a pre-defined query prefix, a select clause derived from the *getSelectClause()* function and a where clause derived from the *getWhereClause()* function. Lines 7-27 define the *getWhereClause()* function in a recursive way. It takes as input the event pattern in the composition plan (Line 7) and finds the root node in the event pattern (Line 8). Then, it investigates the type of the root node: if it is a *Sequence* or *Repetition* operator, the transformation algorithm terminates, currently transformation cannot be applied for *Sequence* or *Repetition* because of the limitations of the underlying query language (CQELS) (Lines 9-10). If the root node is an event service description, a *getSGP()* function creates the Stream Graph Patterns (SGP) in CQELS (Lines 11-12) describing the triple patterns of the observations delivered by the event service, and this SGP is returned

Algorithm 1 Transform event patterns into CQELS queries.

Require: Composition Plan: *comp*, Query Prefix String *prefixStr***Ensure:** CQELS Query String: *queryStr*

```
1: procedure TRANSFORM(comp, prefixStr)
2:   selectClause ← GETSELECTCLAUSE(comp.ep)
3:   whereClause ← GETWHERECLAUSE(comp.ep)
4:   queryStr ← prefixStr + "SELECT" + selectClause + "WHERE" +
   whereClause
5: return queryStr
6: end procedure
Require: Event Pattern: ep
Ensure: Where Clause String: whereClause
7: procedure GETWHERECLAUSE(ep)
8:   root ← GETROOTNODE(ep), whereClause ← ∅
9:   if root ∈  $Op_{seq} \cup Op_{rep}$  then
10:    fail and terminate
11:  else if root ∈ EventServiceDescription then
12:    whereClause ← GETSGP(ep, root)
13:  else if root ∈  $Op_{and}$  then
14:    for subPattern ← GETSUBPATTERNS(ep, root) do
15:      whereClause ← whereClause + GETWHERECLAUSE(subPattern)
16:    end for
17:  else if root ∈  $Op_{or}$  then
18:    for subPattern ← GETSUBPATTERNS(ep, root) do
19:      whereClause ← whereClause + "optional" +
      GETWHERECLAUSE(subPattern)
20:    end for
21:    whereClause ← whereClause + GETBOUNDFILTERS(ep)
22:  end if
23:  if filters ← GETFILTERS(ep) ≠ ∅ then
24:    whereClause ← whereClause + GETFILTERCLAUSE(filters)
25:  end if
26: return "{" + whereClause + "}"
27: end procedure
```

as a (part of the) where clause. If the root node is an *And* or *Or* operator, the algorithm invokes itself on all sub-patterns of the root node and combines the where clauses derived from the sub-patterns (Lines 13-20). In addition, if the root is an *Or* operator, an *OPTIONAL* keyword is inserted for each where clause of the sub-pattern and a bound filter is created indicating at least one of the sub-patterns has bound variables (at least one sub-events occurs, Line 21). If there are filters specified in the event pattern, a *getFilterClause()* function is invoked to add the filter clauses to the where clause (Lines 23-25). Finally, the where clause is returned with a pair of brackets (Line 26). Listing 5 shows the transformation result for event request in Listing 3.

Listing 5. CQELS query example

```
Select * Where {
Graph <http://purl.oclc.org/NET/ssnx/ssn#>
{?ob rdfs:subClassOf ssn:Observation}
Stream <locationStreamURL> [range 5s]
{?locId rdf:type ?ob. ?locId ssn:observedBy ?es4.
 ?locId ssn:observationResult ?result1.
 ?result1 ssn:hasValue ?value1.
 ?value1 ct:hasLongitude ?lon. ?value1 ct:hasLatitude ?lat.
 ?loc ct:hasLongitude ?lon. }
Stream <trafficStreamURL1> [range 5s]
{?seg1Id rdf:type ?ob. ?seg1Id ssn:observedBy ?es1.
 ?seg1Id ssn:observationResult ?result2.
 ?result2 ssn:hasValue ?value2.
 ?value2 ssn:hasQuantityValue ?eta1.}
Stream <trafficStreamURL2> [range 5s] {...}
Stream <trafficStreamURL3> [range 5s] {...} }
```

6 Related Work

Existing event notification services like SAS⁹ and WSN¹⁰ support only publish and subscribe simple and syntactical events. Recent research on semantic event processing endeavor to bring semantics to event specifications. Semantic event specifications have more flexibility and expressiveness compared to syntactical ones[12]. However, most existing event ontologies, (e.g., [14, 17, 18]) lack the ability to describe comprehensive event operators and non-functional constraints. Moreover, they do not discuss the mechanisms of complex event service discovery and reusability.

Reusing event queries/subscriptions is also discussed in many other event based systems, including content-based event overlay networks [2, 3, 10, 8, 11, 6, 13] and CEP query optimisation[1, 15]. In event overlay networks, event subscriptions are reused to facilitate the "downstream replication" and "upstream evaluation" principles (as described in [2]) and reduce the traffic over the network. In event query rewriting and optimisation, sub-queries can be delegated to existing event processing nodes/agents when their patterns match, in order to reduce processing burden of event engines.

Although the above works in event overlay networks and query rewriting share some objectives to our work in terms of improving the network and event processing efficiency, this paper is different because 1) we do not focus on routing algorithms which are central parts of event overlay network research, all nodes in the event service network can host both event producers and consumers and they are visible to all other peers and 2) we do not re-order query operators in a

⁹ Sensor Alert Service: <http://www.ogcnetwork.net/SAS>

¹⁰ Web Service Notification: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

way such that cpu usage and latency can be minimized, which is central to query rewriting techniques. Instead, we develop means to create event service compositions based on the semantic equivalence and reusability of event patterns, and then composition plans are transformed into a set of federated stream reasoning queries, enabling a semantic complex event processing over distributed service networks.

7 Conclusion and Future Directions

In this paper, we have identified several challenges for Smart City applications. We presented ACEIS to automatically discover and integrate heterogeneous sensor data streams and thus addressing the data stream federation challenge. ACEIS receives requirements from users and applications as event request and discovers or composes the relevant data streams to address both functional or non-functional requirements specified in the event requests. The discovery and composition process in ACEIS rely on the CES ontology designed for describing complex event services as extended OWL-S services. Based on the discovery and composition results, ACEIS automatically generates CQELS queries using a query transformation algorithm and registers the queries to a CQELS engine. These queries operate on live semantic data streams produced by various physical as well as virtual sensors to detect complex events for users.

In future, we plan to implement the adaptation component in ACEIS, which can dynamically adapt if any of the underlying data stream stops unexpectedly or has a QoS or QoI update at runtime that violates the quality constraints defined in the original request. The adaptation component should be able to determine efficiently whether a situation is critical and the adaptation is necessary, and if so, take automatic recovery actions accordingly. Defining optimal window size for live stream queries of complex events while considering individual update frequency of the all underlying data streams is also part of our future agenda for the adaptation component. The adaptation component can partially address the reliable information processing challenge by providing an automatic recovery mechanism. We also plan to evaluate the correctness of the query transformation by evaluating query semantics.

References

1. M. Akdere, U. Çetintemel, and N. Tatbul. Plan-based complex event detection across distributed sources. *Proc. VLDB Endow.*, 1(1):66–77, Aug. 2008.
2. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, Aug. 2001.
3. E. Curry. Increasing mom flexibility with portable rule bases. *Internet Computing, IEEE*, 10(6):26–32, Nov 2006.
4. F. Gao, E. Curry, M. Ali, S. Bhiri, and A. Mileo. Qos-aware complex event service composition and optimization using genetic algorithms. In *Proceedings of the*

- 12th International Conference on Service Oriented Computing*, Paris, France, 2014. Springer.
5. F. Gao, E. Curry, and S. Bhiri. Complex Event Service Provision and Composition based on Event Pattern Matchmaking. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, Mumbai, India, 2014. ACM.
 6. S. Hasan, S. O’Riain, and E. Curry. Approximate semantic matching of heterogeneous events. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS ’12, pages 252–263, New York, NY, USA, 2012. ACM.
 7. R. Iyer and L. Kleinrock. Qos control for sensor networks. In *Communications, 2003. ICC ’03. IEEE International Conference on*, volume 1, pages 517–521 vol.1, May 2003.
 8. J. Keeney, D. Roblek, D. Jones, D. Lewis, and D. O’Sullivan. Extending siena to support more expressive and flexible subscriptions. In R. Baldoni, editor, *DEBS*, volume 332 of *ACM International Conference Proceeding Series*, pages 35–46. ACM, 2008.
 9. D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *Proceedings of the 10th international conference on The semantic web - Volume Part I*. Springer-Verlag, 2011.
 10. G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/-subscribe systems. In *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, Middleware ’05, pages 249–269, New York, NY, USA, 2005. Springer-Verlag New York, Inc.
 11. Z. Long, B. Jin, F. Qi, and D. Cao. Reuse strategies in distributed complex event detection. In *Quality Software, 2009. QSIC ’09. 9th International Conference on*, pages 325–330, 2009.
 12. T. Moser, H. Roth, S. Rozsnyai, R. Mordinyi, and S. Biffl. Semantic event correlation using ontologies. In *On the Move to Meaningful Internet Systems: OTM 2009*, pages 1087–1094. Springer, 2009.
 13. G. Mühl. *Large-scale content-based publish-subscribe systems*. PhD thesis, TU Darmstadt, 2002.
 14. A. Sasa and O. Vasilecas. Ontology-based support for complex events. *Electronics and Electrical Engineering*, 113(7):83–88, 2011.
 15. N. P. Schultz-Møller, M. Migliavacca, and P. Pietzuch. Distributed complex event processing with query rewriting. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS ’09, pages 4:1–4:12, New York, NY, USA, 2009. ACM.
 16. R. Stühmer and N. Stojanovic. Large-scale, situation-driven and quality-aware event marketplace: The concept, challenges and opportunities. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, DEBS ’11, pages 403–404, New York, NY, USA, 2011. ACM.
 17. K. Taylor and L. Leidinger. Ontology-driven complex event processing in heterogeneous sensor networks. In *The Semantic Web: Research and Applications*, pages 285–299. Springer, 2011.
 18. K. Teymourian and A. Paschke. Semantic rule-based complex event processing. In *Rule Interchange and Applications*, pages 82–92. Springer, 2009.