GRANT AGREEMENT No 609035
FP7-SMARTCITIES-2013

Real-Time IoT Stream Processing and Large-scale Data Analytics for Smart City Applications

Collaborative Project

D3.3 Knowledge-based Event Detection in Real World Streams

Document Ref.	D3.3
Document Type	Report
Work package	WP3
Lead Contractor	SIE
Author(s)	Dan Puiu (SIE), Nazli Farajidavar (UNIS), Daniel Puschmann (UNIS), Bogdan Serbanescu (SIE), Septimiu Nechifor (SIE)
Contributing Partners	SIE, UNIS
Planned Delivery Date	M30
Actual Delivery Date	M32
Dissemination Level	Public
Status	Delivered.
Version	1.0
Reviewed by	Sefki Kolozali (UNIS) and Stefan Bischof (SAGO)

Executive Summary

The CityPulse project proposes a framework for large-scale data analytics to provide information in (near-) real-time, transform raw data into actionable information, and to enable creating real-time smart city applications.

This document presents the results of the activity A3.4 *Event Detection for Urban Data Streams*. The CityPulse framework exposes two modules to detect the events happening in the cities. The first module uses directly the sensory data sources from the city. The second one can be used to process the data from the social media sources. In the case of CityPulse it is used for processing streams of tweets from Twitter. Both modules are generic and can be used to detect events in different application domains.

At the end of this document we have presented how the event detection component have been used to process the traffic data from Aarhus and the stream of tweet from city of London.

Table of Contents

1 Introduction

With billions of deployed sensors, mobile phones (which can also function as real-time, personal, location-aware sensors), wireless devices, and satellites, today's systems can observe signals from almost every person and place on planet Earth. Building upon observations and interpretations of data, occurrences, and events related to different real-world phenomena and using cyber-social information, it's possible for smart city systems to respond to individual applications and users' needs while still balancing societal context in real-time.

Building such systems requires two fundamental capabilities. First is the ability to combine heterogeneous data and services provided by devices via various communication networks and Internet/Web services — satellite data, weather maps, traffic systems, healthcare records, and financial networks — to obtain spatiotemporal observations. Second is the ability to integrate real-world data/services, crowd intelligence, and existing knowledge on the Web into actionable knowledge that the system uses for automated decision making or that it gives directly to individual users and applications.

Smart city frameworks aimed to optimise this communication and service delivery through more efficient decision making mechanisms while expending as few resources as possible. Hence, online monitoring and cognition of situations is important for both city authorities to leverage the management of city resources and reduce the cost, and for citizens to promote their life quality. Figure 1 illustrates departments that provide public support and management for cities[1].

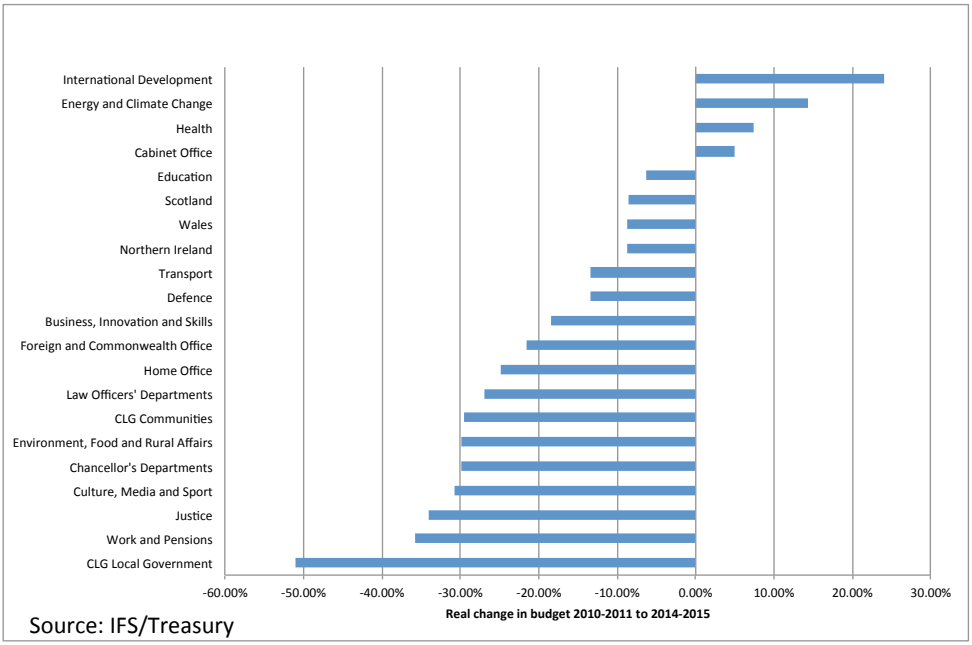

Figure 1: UK governments public spending statistical analyses and budget cut over 2010-2011 to 2014-2015 period for city departments. The figure taken from IFS analysis of the latest Treasury, 2015.

Meanwhile, social media networks such as Twitter[2] provide a platform for citizens to share their opinion in near real time which can serve as an informal service quality assessment based on population probes. This information complements sensor and textual data from conventional sources such as city departments and can be utilised to improve the level of city services. For example, sensors deployed on a road may report reduced speed of vehicles, which can be explained by a procession obstructing traffic.

The CityPulse project[3] uses the IoT data (collected by sensory devices) and social data (collected from social media, such as Twitter) to provide actionable information for various smart city applications. Within the CityPulse project, the event detection component is based on Semantic Web technologies in order to bridge the gap between heterogeneous data sources and modalities. This enables to exchange machine interpretable data with different components: particularly, it receives real-time annotated sensory and social data streams and

[1] https://www.gov.uk/government/publications/public-expenditure-statistical-analyses-2015
[2] https://twitter.com/
[3] www.ict-citypulse.eu

detects the events happening in the city. The event detection component is composed of two main modules: i) event detection from sensory data and ii) event detection from social media data. While the former enables to extract the information from sensors deployed in the city, the latter processing the tweets coming from Twitter.

The remainder of the report is organised as follows. Section 2 presents state of the art methods for event detection for sensory and social media data streams. Section 3 details the proposed methods for event detection component. Section 4 provides the experimental results and Section 5 concludes the report.

2 State of the art

This section details state of the art studies in three parts: i) event-based architectures, ii) machine learning models, and iii) social media analysis.

2.1 Event driven architectures

The event detection goal is to process in near real-time a large volume of observations in order to identify added value events and to provide powerful mechanism for describing structural and temporal relationships between the observations coming from one or several data streams. Real-time complex event processing, which is referred to as Complex Event Processing (CEP), has originated from the work on an object-oriented language designed for system architectures prototyping with powerful event oriented semantics but has seen a sharp increase of interest and usage in the fields of Business Activity Monitoring (BAM), Business Rules Management Systems (BRMS) or Business Intelligence (BI). Although businesses have been conducting near real-time event processing [1], CEP has consolidated itself as a separate topic rather recently. The term itself has been introduced by David Luckham after his work on the Rapide project [2]. Since then, it has been shifted from the analysis of hardware discrete-event systems towards distributed software architectures, and has seen a first detailed description in [3]. CEP has seen a sharp increase in interest and utilisation through new commercial solutions [1], [4], [5], [6].

We could describe a CEP platform as a software solution for developing applications, which are capable of processing information about such happenings, represented as *events*. The goal is to produce value added information represented as *complex events* by exploiting the temporal, causal and structural relations between data streams. As opposed to the earlier event processing solutions, a CEP platform is generic since in the resulting software applications the relations are not hard coded. This increases flexibility and cuts software deployment and maintenance costs.

The guaranteed latencies advertised by most CEP engine providers are in milliseconds or less than milliseconds. These latencies can increase for extremely complex applications but this issue can be overcome with distributed CEP approaches (even with a simple approach involving event partitioning and distributed event processing). A typical CEP engine is usually not suitable for a solution requiring less than a millisecond, such as those for real-time control. Nevertheless, it can easily cope with systems where time constants are larger than tens of milliseconds, where the data transport between the event producer and the CEP engine is not the responsibility of the CEP engine itself and is guaranteed to fulfil these timing requirements.

A typical CEP application consists of an Event Processing Network (EPN), where *raw events* are usually fed through a set of *input adapters* or *event producers* in order to be processed by interconnected *processing agents* or *processing nodes* to obtain *complex events*, which themselves can be the subject of other processing operations. The results of the processing stage are usually sent to *output adapters* or *event consumers*, which can be linked to an Human Machine Interface (HMI) in a form of a monitoring dashboard application, or, to other applications and software components. Depending on the CEP platform, the event processing nodes are performing in *stream-oriented style*, *rule-oriented style* or *imperative-style* as described in the comprehensive work about CEP of Etzion and Niblett [3] and usually can be updated at run-time through a *management interface*.

This ability to handle events has raised interest about CEP not only from the business oriented application providers but also from developers of other types of applications. Balis *et al.* [8] propose a CEP based monitoring solution for grid computing infrastructure. In the manufacturing field, Izaguirre *et al.* [9] introduced a method for the supervision and monitoring of manufacturing systems dealing with heterogeneous event sources. Similarly, Karadgi, *et al.* [10] proposed an event driven framework capable of handling event processing on the device level as well as the enterprise level of a manufacturing execution system. A similar approach presented by Wlatzer *et al.* [11] with some highlights on the HMI, while in [12] Rosales *et al.* focused on the CEP processing of events generated by RFID devices. In [13], [14] or in [15]. Dunkel [16] addressed the issue of event processing for sensor networks and applied event-driven CEP architecture for decision support in traffic management systems in [17]. Another application of CEP in the domain of transportation presented by Terroso-Saenz in [18] and [19].

2.2 Machine Learning Models

In the context of event detection Machine Learning (ML) models can be used to predict the values of some parameters (e.g average speed of cars, or number of cars) after an interval of time (e.g. 15 minutes) based

on current state of the system. By running the CEP rules on the stream of predicted measurements, one can generate predictions about the events that are going to happen.

ML targets development of algorithms, which are capable of learning the sensed behaviour based on the acquired experience. The models resulted in ML are mainly data driven, although a ML expert is critically involved in deciding which type of ML algorithm is the most appropriate for given task and data.

We refer here to Mitchell's assertion that any useful machine learning algorithm must have its own inductive bias, otherwise it is useless [20]. In the same paper it has been stated that "progress towards understanding learning mechanisms depend upon understanding the sources of, and justification for, various biases", beyond the particularities of the training sets. This serves as an explanation for the large number of proposed ML algorithms. Some of the popular algorithms are as follows: artificial neural networks [21], [22], Support Vector Machines [23], Bayesian networks [24], decision trees, association rule learning, and clustering algorithms – which are the conventional methods of ML. We also witness an upsurge interest in hybrid methods that combines multiple learning approaches [25], [26], [27], [28].

The usual workflow for a ML-based process is: *a)* start from raw data; *b)* perform data cleaning and data preprocessing; *c)* apply a ML algorithm to build a model from data. The derived model is subsequently used to process upcoming data, mainly for making predictions (classification, posterior probability estimation, regression) or summarisation.

The data cleaning step is tightly bound to data quality [29], [30]. One has to analyse the data and identify the potential errors in the data set. Numerous methods were proposed, including outlier detection, pattern matching, clustering, and data mining techniques [30]. For example, when merging data sources, one has to remove the duplicate values by means of regular expressions, fuzzy matching, user-defined constraints, filtering, etc. The most effective approach seems to be decomposing and reassembling the data. Here we see CEP as bringing a clear advantage, as through specific EPNs one can express, run and synergistically pipeline such simple steps. Maletic and Marcus ([30], chapter 2) describe the following three phases for a data cleansing process: a) define and determine error types; b) search and identify error instances; c) correct the uncovered errors. The first two approaches are easily to be automated and there are some frameworks imagined for them. The third step is tightly bound to the domain and has to be accordingly developed.

The next step is preparing data for the Machine Learning algorithms. Some of the ML methods specify the data types they are able to process (e.g. neural networks accepts input numerical values solely, and only a few of them allow missing values [31], [32], [33]; some variants of decision trees can perform only classification, etc.). Beside this, it is generally accepted that the good input features may ease the classification phase, up to leading to simple classifiers.

There are three approaches to deal with initial data, all of them are part of feature engineering. The easiest one is to chose among the existing attributes and restrict to the ones that are considered to be relevant for the task at hand. This step may be supported by a domain expert or automatically performed. For example, one can remove correlated attributes which degrade performance of Naive Bayes (the overcounting effect) or for methods based on computing the inverse of a data–derived matrix[4]. Additionally, heuristic methods for feature extraction were developed, e.g. based on gain ratio with respect to the class, chi–squared tests, greedy and evolutionary search etc.

2.3 Social Media Analysis

Typically, a city has many departments such as public safety, urban planning, energy and water, transportation, social programs, and education [46, 47]. Some of the services offered by these departments are dynamic, *e.g.*, transportation services and their behaviour may vary in response to social and cultural events, accidents, and weather conditions. Live cognition of the situation is important for city authorities to leverage the management of city resources. Meanwhile, social media networks such as Twitter and Facebook have developed into a near real-time source of information spanning heterogeneous topics of varying importance. Figure 2 depicts samples of real-world city events (*e.g.*power outages, poor water quality, a procession, and a delay experienced in public transport system) reported directly by citizens. This information complements sensor data or textual data from conventional sources such as city departments and can be utilised to improve the level of city provided services. For example, sensors deployed on a road may report reduced speed of vehicles which can be explained by the procession obstructing traffic.

The need for such social sensing becomes even more crucial in developing countries, where fine-grained sensor data may not be available due to the lack of extensive instrumentation.

[4]Although the Moore–Penrose pseudoinverse may be used, a high condition number shows that the computation is numerically unstable.

(a) Power shutdown

(b) Poor water quality

(c) Traffic jam due to procession

(d) Delay in public transit system

Figure 2: Tweets reporting various concerns about a city spanning power supply, water quality, traffic jams, and public transport delays (© [48]).

To achieve this, the social sensing event extraction task can be formulated through addressing the following research questions: How do we extract city infrastructure related events from twitter? How can we exploit event and location knowledge-bases for event extraction? How well can we extract city traffic events?

Works such as [49] assume the presence of event data sources such as sensor data (*e.g.*, loop detectors) and formal report of events (*e.g.*, eventful[5]) in a city. Such a source of events may not be available in all the cities warranting the need for city event extraction from textual data. On the other hand these formal events when present can serve as ground truth to validate an automated event extraction system. Following Anantharam *et al.* [48], we organize the event extraction from textual data into two categories: open domain (unknown event types) and closed domain (known event types). Additionally, the textual data can be of either formal or informal nature. Where the former refers to the grammatical text such as news documents and the latter addresses the user-generated content with no overt structure which might contain a lot of slang and non-standard abbreviations and notations.

In formal text analysis context, Liu *et al.* [50] propose to alleviate information overload in daily news by extracting key entity and significant event of news documents. A bipartite graph is induced based on the entities and their associations to documents using mutual reinforcement principle capturing salient entities and the documents with salient entities to rank the news events. Extraction of local events from blog entries has been carried out by [51]. Use of lightweight patterns to extract global crisis events from news text is presented in [52]. Event extraction in the context of detecting infectious disease outbreak was done by [53]. The event schema consists of date range, geo-location, disease name, organism type and number affected by the disease, and the organism survival information. The event extraction is done by finite-state pattern matching on the tokenized input text.

Event extraction from informal text is recently addressed in literature [54, 55, 48]. In [54], the authors proposed the use of temporal (volume changes), social (replies, broadcast), topical (coherence of clusters), and twitter-centric (multi-word hashtags) features to train a classifier that performs better than the baseline. Ritter *et al.* [55], demonstrated that building a calendar of significant events is feasible using twitter stream using an unsupervised approach to process tweets and extract event types such as sports, concert, protests, politics, TV, and religion. The approach utilised the Latent Drichlet Allocations (LDA) method to model each entity in terms of a mixture of event types and each event type in terms of a mixture of entities. Following the same trend, Wang *et al.* [56] used tweets for predicting hit and run crimes. A latent topic based model constructed over semantic role labels [57] of events from tweets. A generalised linear regression model is used to capture the association between topics and crimes from a training dataset. Lampos and Cristianini [58] proposed to use an optimised feature selection approach coupled with regression to estimate the intensity of events based on event markers. An evaluation based on ground truth from rain gauges is used for validation. They also extended the evaluation to identify Influenza Like Illness and compare it with the data from Health Protection Agency [6].

Most recently, Anantharam *et al.* [48] developed an automatic twitter-training data creation process to train a ground-truth annotation model by utilising officially reported events [7] and locations [8] knowledge-bases. In the next stage, they used this annotated data to train a CRF-based event extraction model specific to twitter

[5] http://eventful.com/
[6] http://www.hpa.org.uk/
[7] http://511.org
[8] https://www.openstreetmap.org/

textual domain. Despite a successful ground-truth annotations system the Conditional Random Field (CRF)-based event extraction unit of [48] has some limitations which is as follows: (1) since the model is trained on a limited volume of training data (tweets collected in a 3-month time interval) it lacks generalisability. Due to training the CRF with the officially reported ground-truth events of a limited time period, the model fails in detecting future events of a new nature. (2) although the location terms have been extracted within the model but these information had not been used for obtaining the events' location. Instead, the authors suggested to use the tweet's geo-location tag (the location where the users tweet the events) which does not necessary agree with actual event's location.

These limitations motivated our work. Utilising the ground-truth annotation module of [48], instead of training a CRF we have adopted the Convolutional Neural Network (CNN) model of [59] for extracting both the event and location terms. Unlike Anantharam *et al.*'s[48] model, our proposed model is trained on a more generic categorical training data and hence generalises well to future events. While various neural network architectures [60, 61, 62] have been proposed in literature and their performance are investigated for twitter sentiment classification, to the best of our knowledge, this is the first time the CNN text analysis is utilised for city even extraction.

3 Event Detection

3.1 Event detection from streaming data

Figure 3 depicts the architecture of the proposed framework which allows event detection for sensory data streams. It provides a set of generic tools which can be used for detecting an event. In order to detect a certain event, the domain expert, has to deploy a piece of code called event detection node. These nodes contain the event detection logic, which represent the pattern that has to be identified during raw data processing.

A domain expert, with some help from an application developer, can use two configuration interfaces displayed by the framework in order to perform the following activities:

- Describe how to access the streams endpoints and how to interpret the received messages on a design time phase;
- Register the data streams from the city by providing their descriptions;
- Develop the event detection nodes on a design time phase;
- Deploy the event detection nodes for various stream combinations.

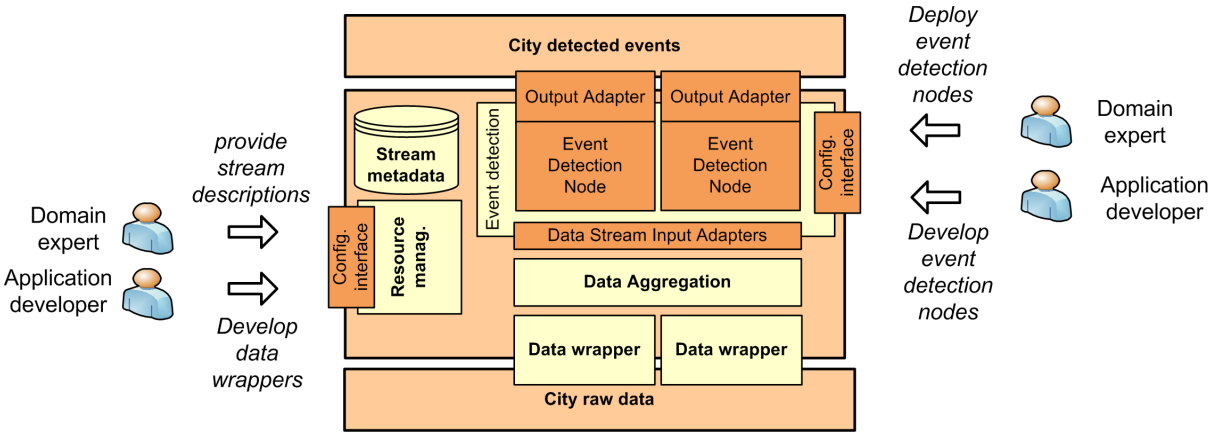

Figure 3: Framework arhitecture

3.1.1 Data streams input adapters for event detection

The role of the data stream input adapters in the event detection component is to connect to the appropriate routing keys of the data bus in order to fetch the required data needed by the event detection nodes. For each raw data stream (source) a new input adapter has to be deployed in order to fetch the data. The deployment of input adapters is done automatically when a data source (semantically annotated or aggregated observation) is going to be needed by at least one event detection node. Having the description of each data stream stored into the stream metadata store, the input adapters are deployed directly without having to develop supplementary pieces of code, which allow the conversion from the N3 format to the formats needed by the CEP engine.

During the deployment phase of the event detection nodes, the domain expert has to specify the UUIDs of the streams which have to be processed. Having the UUIDs, the event detection node can determine if an already deployed event detection input adapter, required by a previously deployed event detection node, can be reused or a new one has to be instantiated.

In order to deploy a new input adapter the event detection node, one first has to get the description of the stream. This information is obtained by generating a request, containing the UUID of the stream, to the resource management. By decoding the response, the event detection node knows the routing key of the data bus where the stream observations are going to be published and the name/types for the fields of the messages. After that an input adapter is deployed, by creating a new data bus client for the specified routing

key. Every time a message is delivered on the data bus routing key, the client decodes it, using the known name/types of fields. At the end the message is converted and delivered to the CEP engine.

3.1.2 Event detection component

The event detection component represents a generic environment where the domain expert can deploy various event detection nodes. The component was developed as a wrapper over the Esper complex event processing engine. The Esper engine allows the user to define custom made Event Processing Networks (EPN), which can be used for detecting various patterns in the streams of data.

An alternative to our proposal would be to use directly the Esper engine. In this way an application developer has to design the EPN and to develop the pieces of code responsible for fetching the data from the various data sources. Furthermore, the application developer has to have increased attention when dealing with heterogeneous data sources and when having to use the same data source for different EPNs. The main drawback of this approach is that it offers little reusability of the developed pieces of code.

The integration we are proposing has the following advantages:

- Once a data wrapper has been developed it can be reused very easy for other data sources of the same type;
- The sensor measurements are delivered to the event detection component following the same standard format (if one has to deploy two temperature sensor which have to different protocols and data formats he will have to develop two data wrappers but this is not going to be visible at event detection level);
- Different persons can be involved to develop the data wrappers and the event detection nodes;
- During the deployment phase of the event detection nodes, the domain expert has only to configure the binding to the data streams by proving the UUIDs;
- The same event detection node can be deployed for different combinations of inputs streams.

3.1.3 Event detection nodes

An event detection node represents the generic event processing logic used for detecting special situations from the raw or aggregated data such as traffic incidents or parking status incidents. Since the event detection node is generic (in this case it does not contain any information about the specific data sources which are going to be used), the domain expert can select the raw or annotated data sources which represent the inputs for the node. Also the domain expert can configure these nodes by modifying the input parameters if necessary. The event detection node can be seen as a black box with inputs (input streams and configuration parameters) and having as output the detected events (Figure 4).

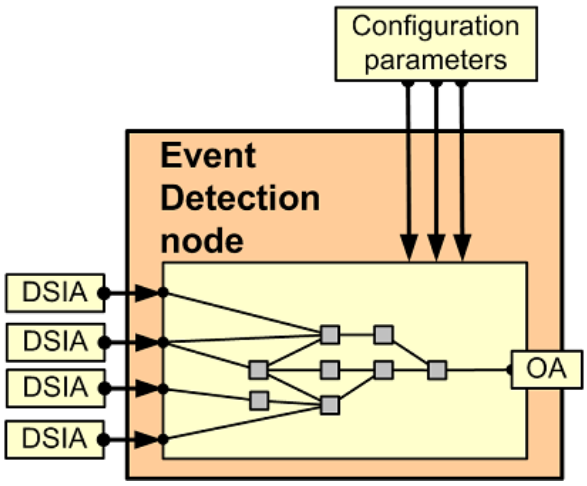

Figure 4: Event detection node

The generic event detection nodes are represented by Java classes. The objects resulting after instantiation (when all the configuration parameters and the binding to the event streams have to be provided) are deployed into the event detection component via the configuration interface. In order to develop a new event detection node the application developer has to extend the abstract class *EventDetectionNode* (see Figure 5). For the sake of simplicity, Figure 5 shows only the fields and methods relevant to the application developer. Methods that implement default behaviour are hidden. Those are used by the event detection component for deploying the appropriate EPN and for adding the corresponding input adapters.

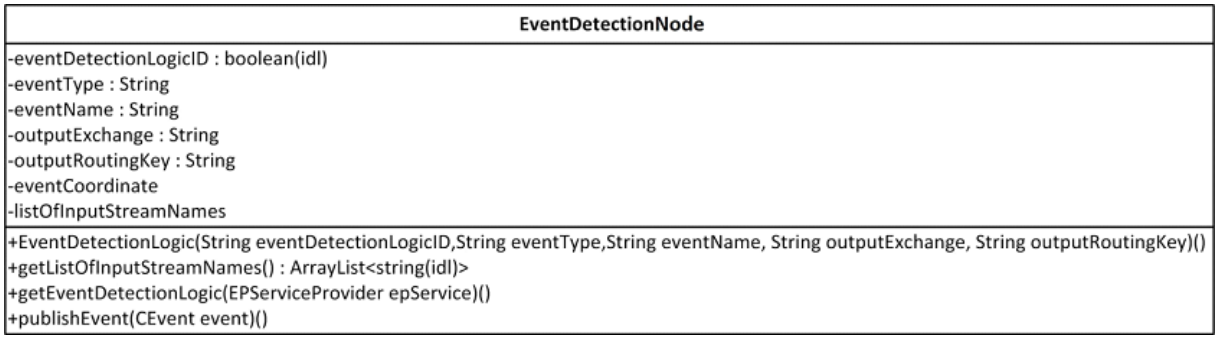

Figure 5: Event detection logic UML class diagram

The application developer has to implement the *getListOfInputStreamNames()* and *getEventDetectionLogic()* methods. The first method returns an array list of strings, where each string represents the name of an input adapter. This method is called during the deployment of the node in order to check if the domain expert has provided the UUIDs of the streams which have to be bind to the inputs.

The second method,*getEventDetectionLogic()*, is used by the application developer to define the event detection pattern. The parameter *epService* represents the handle to a running CEP engine. In this way, the Esper EPL can be used to define the event detection logic. The domain expert defines, via the constructor, the following fields:

- eventDetectionLogicID: the name of the event detection node;
- eventType: the category of the event, which can be for example: “traffic”, “weather”, etc.;
- eventName: the name of the event, which can be for example: “traffic jam”, ”high temperature”, etc.;
- outputRoutingKey: the data bus routing key where the events are going to be published.

An event detection node can be adjusted via the configuration parameters. In most common situation these parameters represents various threshold embedded into the CEP statements. In order to achieve this, the application developer has to create a new constructor (with the needed parameters) and within its implementation call the *EventDetectionLogic* constructor. The event processing network, which is defined with the method *getEventDetectionLogic()*, generates as output an complex event when the detection conditions are meet. The application developer can use the method *publishEvent()* for publishing the event on the data bus. During the deployment phase, the domain expert will have to provide values for the above mentioned fields and parameters. In addition to these the binding of the event detection node and the required inputs has to be defined.

The generic event detection mechanism presented in this section are used to process the aggregated or semantic annotated data and to detect the event when it happens. There are a lot of situations when it is useful to predict the apparition of an event with some time before it is produced. In order to achieve these, ML prediction models can be used and have to be integrated into an event detection node. We have used Weka [63] to train the various prediction models. After that, using the Weka API [64], the model can be extracted and injected into the event detection node.

3.1.4 Event detection output adapters

The role of the data stream output adapters, within the event detection framework, is to translate the output of an event detection node into a standard format for detected events. Also, this component is responsible to

publish the detected events on the message bus. The output adapters' deployment is done automatically for each event detection node. All detected events contain the following fields:

- eventID: unique identification number;
- eventType: the category of the event (e.g. transportationEvent);
- eventName: the name of the event (e.g. trafficJam, parkingStatus);
- eventTime: the moment when the event was detected;
- eventCoordinate: the location of the event;
- eventLevel: the gravity of the event. Events can be categorised, for example, into four categories after level: 0 - not present, 1 - small, 2 - medium, and 3 - serious.

3.1.5 Event detection work flow

The Figure 6 depicts the workflow which has to be followed for deploying the data wrappers and the event detection nodes.

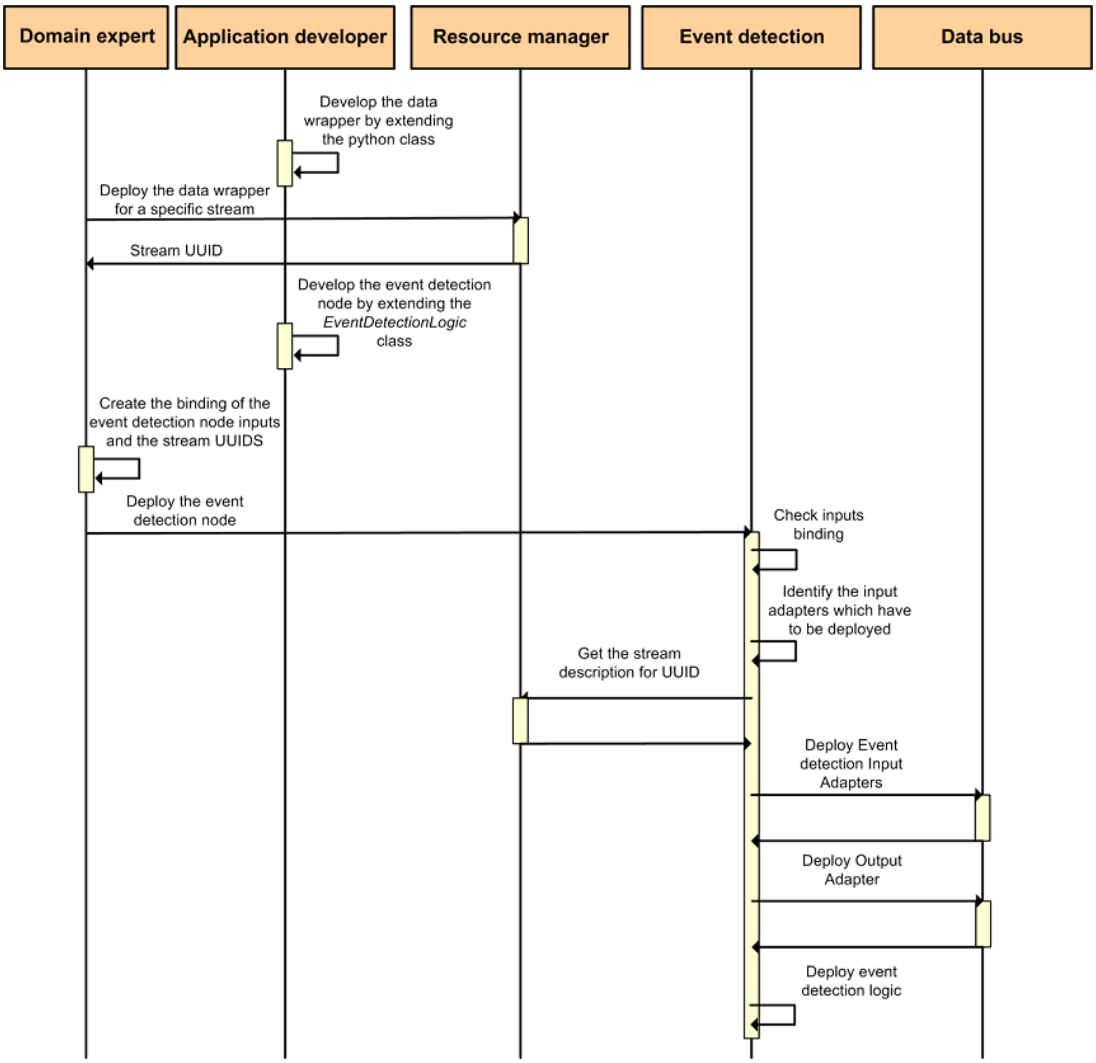

Figure 6: Data wrappers and the event detection nodes deployment workflow

An initial step to build an event detection workflow is to develop the data wrapper. This part is implemented by application developer, extending the Python classes. A domain expert, using the Resource management API,

deploys the implemented data wrapper for the specific data stream; and also obtains the stream identification (UUID) after the data wrapper deployment. On the other hand, the application developer has to develop also the event detection node by extending the *EventDetectionNode* class. In order to deploy the created event detection node, the domain expert has to correlate its inputs and the stream UUIDs. Together with event detection node deployment, the event detection component will add a new input adapter for the data bus. This implies first to check the input binding, identify the input adapters which have to be deployed and obtaining the stream's description based on the UUID by requesting the resource management. Furthermore, the event detection component deploys alsol the output adapter on data bus and the last step is to deploy the event detection node, based on which the input data is processed.

3.2 Event detection from social streams

The second module exposed by the Event detection component is used to process the Social Media streams. The module analyses and annotates large-scale real-time Twitter data streams. A dedicated Data wrapper is used to connect to the Twitter stream API and collect the data in the form of tweets. The Data wrapper uses the Google-translate API to automatically detect the source language and translate the tweets to English to facilitate the Natural Language processing step. This in fact enables the application developer to fetch data from any area of the globe (e.g. the area surrounding a certain city).

The Social Media Event detection component reads a sequence of words in the sentence (Tweet), and passes it to a Natural Language Processing (NLP) unit (see Figure 7).

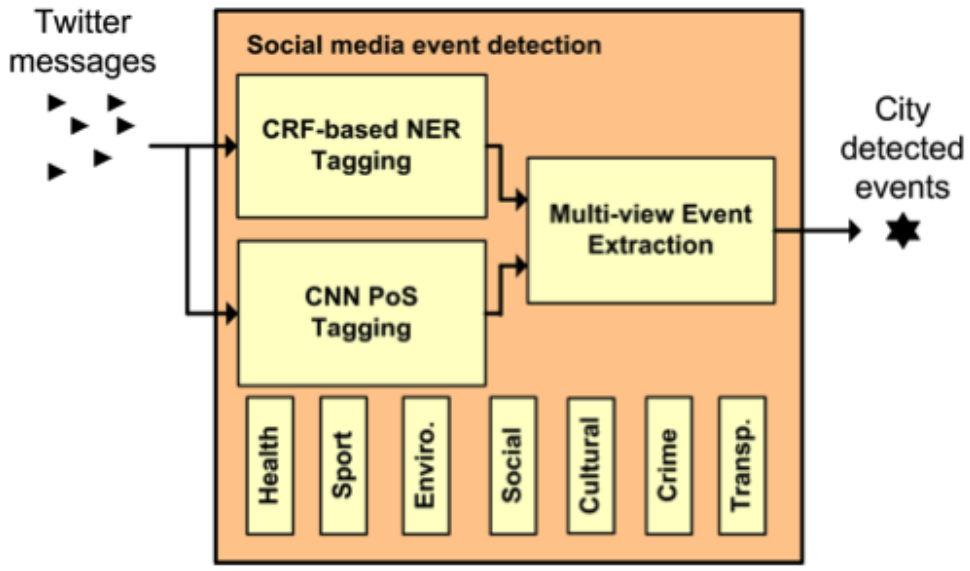

Figure 7: Social Media stream processing and event detection component

The processing unit is composed of three sub-components: a Conditional Random Field Name Entity Recognition [65, 66], a Convolutional Neural Network for deep learning for Part of Speech tagging, and a multi-view event extraction.

During the design time, the internal sub-components are trained with a large corpus of Wikipedia documents and historical Twitter data to guarantee a generalisable Natural Language Processing model. The Conditional Random Field Name Entity Recognition component assigns event tags to the words in a Tweet given an event-categories set (TransportationEvent, EnvironmentalEvent, CulturalEvent, SocialEvent, SportEvent, HealthEvent, CriminalEvent), which is tailored for city related events. The categories have been defined generic enough to cover future sub-categorical event assignments (*e.g.*traffic and weather forecast events can be perceived as sub-categories of TransportationEvent and EnvironmentalEvent categories, respectively.) and they will be available to third party developers for adoption and utilisation for new scenarios. Additionally, their respective event vocabularies are adaptive and extendible to future events.

During the run time, the Conditional Random Field Name Entity Recognition component assigns event tags to the words in a Tweet from the event categories set. Simultaneously, the trained Convolutional Neural Network [59] component generates Part-of-Speech tags for atomic elements of the Tweet. The obtained two

views of the data (Conditional Random Field Name Entity Recognition view and Convolutional Neural Network Part of Speech view) are then fed into a novel multi-view event extraction component, where the obtained tags of the two views are mutually validated and scored for a final sentence-level inference. An example of sentence level inference is in the case of tweets such as "seeing someone being given a parking ticket" where individual words "parking" and "ticket" can belong to Transportation and Cultural events categories respectively while considering the sentence grammar can clear up this confusion and assign the tweet to Transportation category. The real-time extracted city events are then used by Real-time Adaptive Urban Reasoning component to obtain a more comprehensive interpretation of the city events. The extracted knowledge is utilised to complement the sensor stream information extraction and allows obtaining of a more detailed interpretation of the city events when complementary citizen sensory data can be extracted via social media processing.

4 Implementation and Experimental Results

This section details the experimental results of the event detection component for near real-time sensory and social media data streams.

4.1 Event detection from streaming data

In this section we demonstrate how to use the proposed framework to detect traffic events by processing sensory data streams provided by the city of Aarhus, Denmark. For the evaluation we used two different kinds of data streams: traffic and parking data streams.

Traffic data streams consist of two spatial-separated observing stations. A simple exemplification of a traffic sensor observation is depicted in Figure 8. Each observation contains sensor readings of the number of vehicles and average speed regarding the vehicles that have passed between two sensor points. In total 449 of these pairs stations are available. These sensors provide near real-time observations to the system with a frequency of 5 minutes time interval. Traffic data stream is a composed data stream, meaning that we recieve multiple observations in one single request. Therefore, we have used a composed data wrapper to fetch data. The data source is a REST service that delivers JSON encoded data. In addition the service used pagination like iteration over a subset of all observations. A specific Sensor Connection and a Splitter were implemented, whereas a default implementation of a JSON parser could be used.

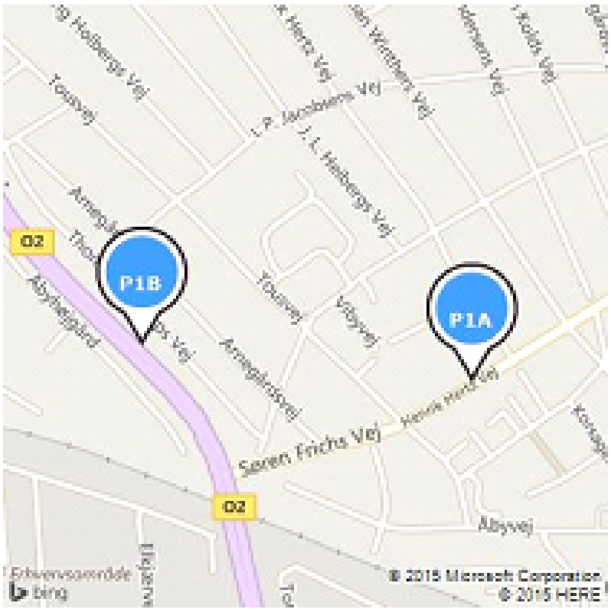

Figure 8: Traffic sensor observing traffic from point P1A going to P1B

Parking data streams provides real-time information regarding the capacity of parking garages in Aarhus. The measurements encompasses the number of total parking spaces and the number of vehicles currently in the parking garage. In total, there are 13 parking garages are available in the city of Aarhus. The goal of the event detection component is to process the sensory data for detecting the moments when traffic jams occurs or when the number of parking places from garages significantly drops in a very short period of time. A special characteristic in the parking stream is the fact that the upper bound for the number of vehicles in the garage depends on the number of total spaces, which changes over time. Reasons for such changes can be the addition of an extra level in the garage or the blocking due to maintenance work. This feature of the data stream has been taken into account in the development of the data wrapper. Figure 9 shows a sample of an annotated parking data stream observation. The explanation of the triples represented in the given graph are as follows (i.e. top-down): observation value, observation measurement and sampling time, the reference to observing sensor and the reference to the observed property.

```
@prefix ct: <http://ict-citypulse.eu/city#> .
@prefix muo: <http://purl.oclc.org/NET/muo/muo#> .
@prefix qoi: <http://purl.oclc.org/NET/UASO/qoi#> .
@prefix sao: <http://purl.oclc.org/NET/UNIS/sao/sao#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix tl: <http://purl.org/NET/c4dm/timeline.owl#> .

<http://ict-citypulse.eu/sensor/Navitas/Observation-177e780f-c4f7-4a3b-b2b5-134822a52a9e> a
sao:Point ;
   sao:value "Navitas" ;
   ssn:observationResultTime <http://ict-citypulse.eu/sensor/Navitas/ResultTime-6495778b-5a1f-
4e58-b6a8-b707cf3a523a> ;
   ssn:observationSamplingTime <http://ict-citypulse.eu/sensor/Navitas/SamplingTime-6495778b-
5a1f-4e58-b6a8-b707cf3a523a> ;
   ssn:observedBy <http://ict-citypulse.eu/SensorID-c0d94b2a-c7b5-54a9-8169-598671eb2b62> ;
   ssn:observedProperty ct:Garagecode-c0d94b2a-c7b5-54a9-8169-598671eb2b62 .

<http://ict-citypulse.eu/sensor/Navitas/Observation-4a844a9e-2dbd-4c3b-a401-bb4578fe3e41> a
sao:Point ;
   sao:value "449" ;
   ssn:observationResultTime <http://ict-citypulse.eu/sensor/Navitas/ResultTime-6495778b-5a1f-
4e58-b6a8-b707cf3a523a> ;
   ssn:observationSamplingTime <http://ict-citypulse.eu/sensor/Navitas/SamplingTime-6495778b-
5a1f-4e58-b6a8-b707cf3a523a> ;
   ssn:observedBy <http://ict-citypulse.eu/SensorID-c0d94b2a-c7b5-54a9-8169-598671eb2b62> ;
   ssn:observedProperty ct:TotalSpaces-c0d94b2a-c7b5-54a9-8169-598671eb2b62 .

<http://ict-citypulse.eu/sensor/Navitas/Observation-6495778b-5a1f-4e58-b6a8-b707cf3a523a> a
sao:Point ;
   sao:hasUnitOfMeasurement <http://ict-citypulse.eu/unit:number-of-vehicle-per-1min> ;
   sao:value "201" ;
   ssn:observationResultTime <http://ict-citypulse.eu/sensor/Navitas/ResultTime-6495778b-5a1f-
4e58-b6a8-b707cf3a523a> ;
   ssn:observationSamplingTime <http://ict-citypulse.eu/sensor/Navitas/SamplingTime-6495778b-
5a1f-4e58-b6a8-b707cf3a523a> ;
   ssn:observedBy <http://ict-citypulse.eu/SensorID-c0d94b2a-c7b5-54a9-8169-598671eb2b62> ;
   ssn:observedProperty ct:ParkingVehicleCount-c0d94b2a-c7b5-54a9-8169-598671eb2b62 .

<http://ict-citypulse.eu/unit:number-of-vehicle-per-1min> a muo:UnitOfMeasurement .

<http://ict-citypulse.eu/sensor/Navitas/ResultTime-6495778b-5a1f-4e58-b6a8-b707cf3a523a> a
tl:Instant ;
   tl:at "2015-09-02T13:46:29" .

<http://ict-citypulse.eu/sensor/Navitas/SamplingTime-6495778b-5a1f-4e58-b6a8-b707cf3a523a> a
tl:Instant ;
   tl:at "2015-09-02T13:46:29" .
```

Figure 9: Sample annotated observation from the parking data stream.

4.1.1 Traffic and parking event detection nodes

The traffic jam event detection node: can be used for processing the data generated by the traffic sensors with the scope of detecting the moments when a traffic jam occurs. The observations generated by the traffic sensors deployed in Aarhus have two main features which can be used for detecting the traffic jam condition, as follows:

- averageSpeed: the average speed of the cars passing two spatial-separated observing stations as shown in Figure 7 during the last 5 minutes;
- vehicleCount: the number of cars passing through the sensing range of the sensor in the last 5 minutes.

A traffic jam event is detected when a large number of cars pass through the sensing area of the sensor travelling at a low speed for a longer period of time. When the traffic from that area becomes normal than a new traffic jam event is generated but with level zero, signalling that traffic in that area has become normal after a traffic jam. Figure 10 depicts the main blocks of the traffic jam event detection node. The input is represented by one traffic stream from the city and the configuration parameters are:

- averageSpeedTreshold: the maximum value of the average speed for considering a traffic congestion;

- vehicleCountThreshold: the minimum value of the number of cars for considering a traffic congestion;
- trafficCongestionLength: the minimum time interval for considering a traffic jam.

The same event detection logic can be used for extracting traffic events from the aggregated data or from the semantic annotated data. The data aggregation component of the CityPulse framework is configured to aggregate the stream of numeric observation and to generate an aggregated event when the observed value exceeds a threshold. As a result of this, it is highly possible that not all the features will have a significant modification at the same time (e.g. in case of traffic data, to have on the same time a significant change for average speed and vehicle count). Based on this the aggregation component generates aggregated events for each feature. From the event detection perspective, this implies the development of a CEP module used for aligning the aggregation events for traffic. The temporal alignment of the aggregated average speed and vehicle count events is achieved using the statements from the listing 1.

```
insert into TrafficStream select averageSpeedEvent.averageSpeed,
    numberOfCarsEvent.numberOfCars from AverageSpeedAggregatedStream.std:
    lastevent() as averageSpeedEvent, NumberOfCarsAggregatedStream.std:
    lastevent() as numberOfCarsEvent
```

Listing 1: Temporal alignment of the aggregated average speed and vehicle count events.

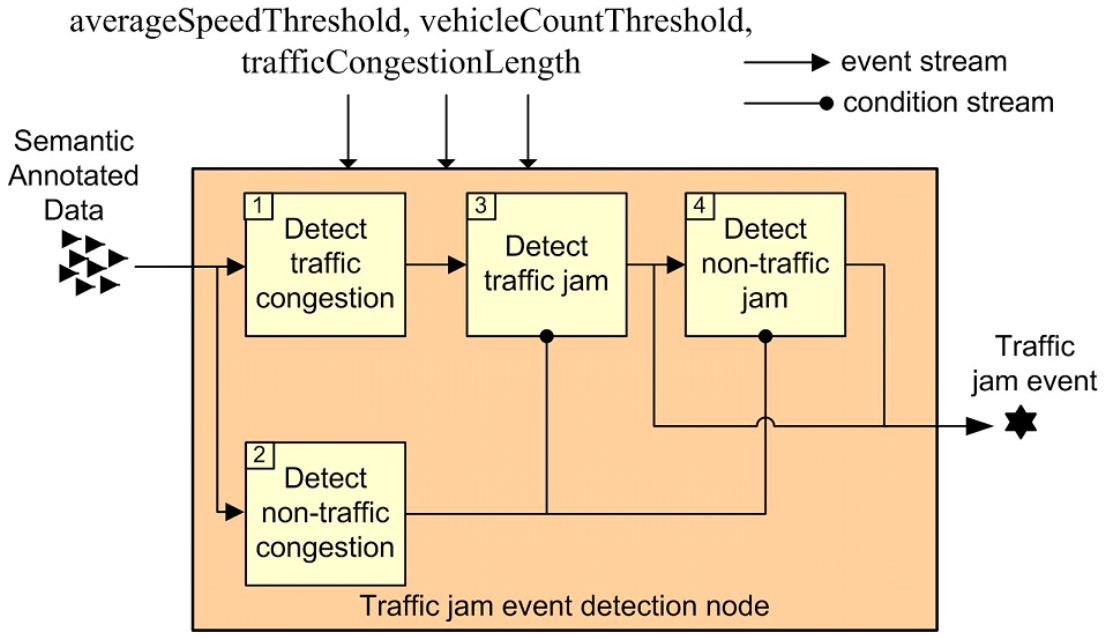

Figure 10: Traffic Jam event detection node

The output of the node will also be a RDF message that will look like the message from Figure 11.

Listing 2 presents the CEP statement that is used to detect a traffic congestion. The statement inserts into the TrafficCongestionStream all the traffic events which have the average speed smaller than *averageSpeedThreshold* and the number of cars bigger than *vehicleCountThreshold*.

```
insert into TrafficCongestionStream select * from TrafficStream where (
    averageSpeed <= (averageSpeedThreshold) and vehicleCount >= (
    vehicleCountThreshold))
```

Listing 2: Critical situation events detection

Non-traffic congestion events are generated when the above mentioned pattern is no longer met (Listing 3).

```
@prefix geo:   <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix sao:   <http://purl.oclc.org/NET/UNIS/sao/sao#> .
@prefix tl:    <http://purl.org/NET/c4dm/timeline.owl#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix prov:  <http://www.w3.org/ns/prov#> .
@prefix ec:    <http://purl.oclc.org/NET/UNIS/sao/ec#> .

sao:4b2e19f1-c135-41d6-a854-755ba7915c3f
        a               ec:TrafficJam ;
        ec:hasSource    "80043a6c-3bdb-5e47-9199-151fb8cfd619" ;
        sao:hasLevel    "1"^^xsd:long ;
        sao:hasLocation [ a        geo:Instant ;
                          geo:lat  "56.19013109480471"^^xsd:double ;
                          geo:lon  "10.218501513474507"^^xsd:double
                        ] ;
        sao:hasType     ec:TransportationEvent ;
        tl:time         "2016-02-15T09:20:41.722Z"^^xsd:dateTime .
```

Figure 11: Traffic Jam event

```
insert into NonTrafficCongestionStream select * from TrafficStream where (
    averageSpeed > (averageSpeedThreshold) and vehicleCount < (
    vehicleCountThreshold))
```

Listing 3: Non-critical situation events detection

A traffic jam event is generated using the query from Listing 4, when a traffic congestion event is not followed by a non traffic congestion event in a time interval shorter than *trafficCongestionLength*.

```
insert into TrafficJamStream select * from pattern [every
    TrafficCongestionStream -> (timer:interval(trafficCongestionLength sec)
    and not NonTrafficCongestionStream)];
```

Listing 4: Traffic jam events detection

The normal traffic event is generated every time when a traffic congestion event is followed by a not traffic congestion event (Listing 5).

```
insert into NormalTrafficStream select * from pattern [every
    TrafficJamStream -> NonTrafficCongestionStream]
```

Listing 5: Traffic jam finalization events detection

Using the Esper API, the events generated on the streams *TrafficJamStream* and *NormalTrafficStream* are detected. Every time an event is received the method *publishEvent()* of the *EventDetectionNode* class is used for delivering the event via the data bus. All the statements presented above are used to implement the method *getEventDetectionLogic()*. In this case the method *getListOfInputStreamNames()* is implemented to return the string trafficDataSource.

Parking event detection node: the scope of the parking event detection node is to detect the changes of the status of the parking garages. More precisely it generates an event when the parking garage gets almost full or when there is a parking places fast vacancy modification (in other words when a lot of vehicles enter the parking garage in a very short period of time). The observations generated by Aarhus parking garages sensors have two main features:

- numberOfCars: represents the current number of cars parked in the garage;
- parkingCapacity: represents the total number of parking places from the garage.

Figure 12 depicts the main logic of the parking event detection node. The input is represented by one parking data stream and the configuration parameters are as follows:

- parkingNearlyFullTreshold: the occupancy percentage of the parking in order to generate the event;
- ocupancyChangeRateTreshold: the rate of car entering into the garage in order to generate the event;
- parkingMonitoringInterval: the length of the time interval for which the occupancy change rate is computed.

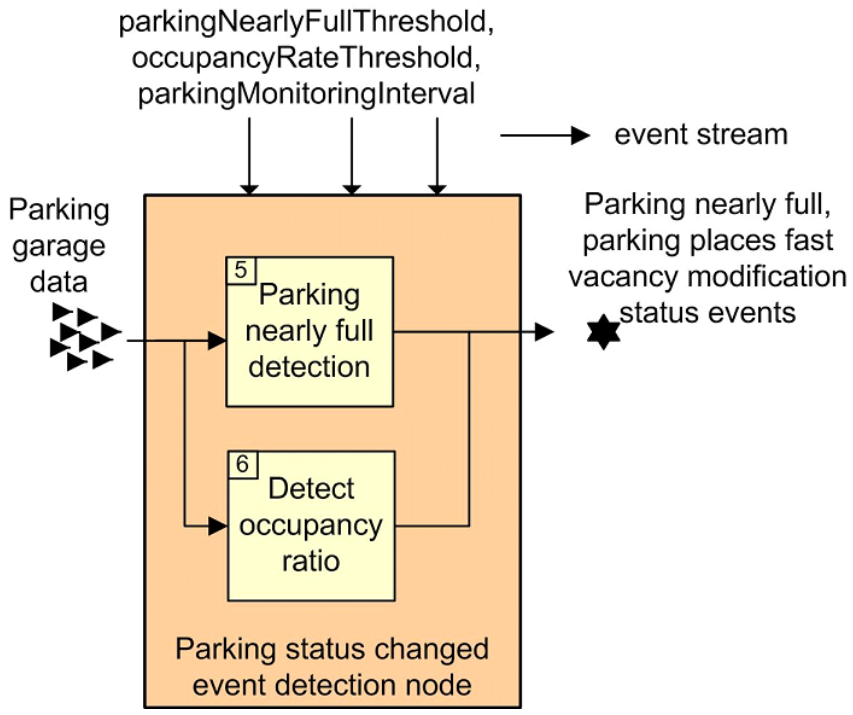

Figure 12: Parking status changed event detection node

The output of the node is also a RDF message that has the structure depicted in Figure 13.
The statement from Listing 6 generates an event when the parking garage is nearly full.

```
insert into ParkingGarageStatusStream select * from ParkingGarageStream
    where (numberOfCars / parkingCapacity >= parkingNearlyFullTreshold)
```

Listing 6: Parking nearly full events detection

The statement from Listing 7 computes first the minimum and the maximum number of cars which have been in the garage in the last *parkingMonitoringInterval* seconds. Then the statement determines with what percentage the parking occupancy have been modified by dividing the difference between the maximum and the minimum values to the total capacity of the garage. If the percentage is bigger than the *ocupancyChangeRateTreshold*, then a parking place fast vacancy modification event is generated.

```
insert into ParkingGarageStatusStream select * from ParkingGarageStream.win
    :time(parkingMonitoringInterval sec) having (max(ParkingGarageStream.
    numberOfCars)-min(ParkingGarageStream.numberOfCars)/ ParkingGarageStream
    .parkingCapacity) > ocupancyChangeRateTreshold)
```

Listing 7: Detect parking occupancy rate events

```
@prefix geo:   <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix sao:   <http://purl.oclc.org/NET/UNIS/sao/sao#> .
@prefix tl:    <http://purl.org/NET/c4dm/timeline.owl#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix prov:  <http://www.w3.org/ns/prov#> .
@prefix ec:    <http://purl.oclc.org/NET/UNIS/sao/ec#> .

sao:9386a9b8-35ea-4e3b-8dc0-364686e2e171
        a               ec:PublicParking ;
        ec:hasSource    "4a838c4b-30d0-5fb4-b3b5-16d6c5c4ff9f" ;
        sao:hasLevel    "2"^^xsd:long ;
        sao:hasLocation [ a         geo:Instant ;
                          geo:lat   "56.16184"^^xsd:double ;
                          geo:lon   "10.21284"^^xsd:double
                        ] ;
        sao:hasType     ec:TransportationEvent ;
        tl:time         "2016-02-15T08:38:10.016Z"^^xsd:dateTime .
```

Figure 13: Parking status changed RDF event message

As in previous case the *sendEvent()* method is called when an event is received from the *ParkingGarageStatusStream* and the method *getListOfInputStreamNames()* is implemented to return the string parkingGarageDataSource.

4.1.2 Noise Level event detection node

The scope of the noise level event detection node is to detect the changes of the status of the level of noise in the city. To be more precise, is generates events when the number of cars on the street multiplied with the average speed of the cars is higher or lower than some thresholds. In order to detect the level of noise the node uses the observations generated by the traffic sensors deployed in Aarhus. It needs both of its features to detect the pollution level:

- vehicleCount: the number of cars passing through the sensing range of the sensor in the last 5 minutes.
- averageSpeed: the average speed of the cars passing two spatial-separated observing stations during the last 5 minutes;

Figure 14 depicts the main logic of the noise level event detection node. The input in represented by one traffic stream from the city of Aarhus and the configuration parameters are:

- noiseLevelThresholdLow: the lower threhsold representing the minimum noise level.
- noiseLevelThresholdHigh: the higher threhsold representing the maximum noise level.

A Low level(level 0) noise event is detected when the number of cars passing through the sensing area of the sensor multiplyed by their average speed is lower than *noiseLevelThresholdLow*. When the number of cars multiplyed by their average speed in inbetween the *noiseLevelThresholdLow* and *noiseLevelThresholdHigh* thresholds the node generates a Medium level(level 1) noise event; and when the number of cars multiplyed by their average speed is above the upper threshold *noiseLevelThresholdHigh* the node generates a High level(level 2) noise event.

The semantic Annotated data(Traffic Data) that is set as an input for the Noise Level event detection node is the same as in the case of the Traffic Jam event detection node and is represented as a RDF message, with the format depicted in Figure 9. The output of the node is also a RDF message depicted in Figure 15.

Listing 8, 9 and 10 present the CEP statements used to detect the levels of noise. There is a total of 3 statements that detect low, medium and high levels of noise.

```
insert into AarhusNoiseLevelStreamLow select * from TrafficStream where (
    StreetVehicleCount * AverageSpeed <= pollutionThresholdLow)
```

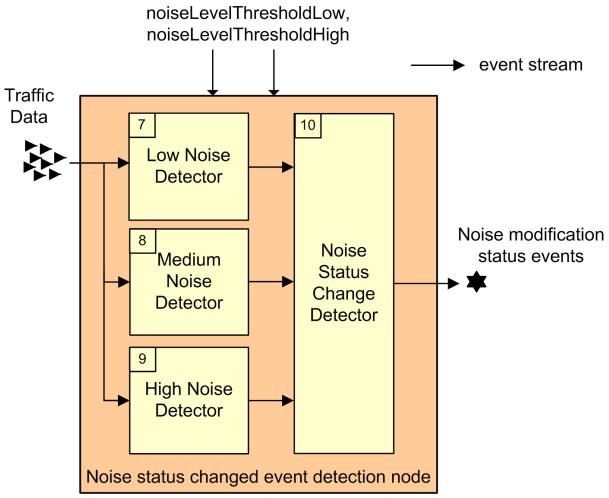

Figure 14: Noise event detection node

```
@prefix geo:   <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix sao:   <http://purl.oclc.org/NET/UNIS/sao/sao#> .
@prefix tl:    <http://purl.org/NET/c4dm/timeline.owl#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix prov:  <http://www.w3.org/ns/prov#> .
@prefix ec:    <http://purl.oclc.org/NET/UNIS/sao/ec#> .

sao:703c2855-0dcf-42bb-aab8-66f05f68fa95
        a               ec:AarhusNoiseLevel ;
        ec:hasSource    "SENSOR_e6232b17-3982-592f-8fc3-5e3c613421aa" ;
        sao:hasLevel    "2"^^xsd:long ;
        sao:hasLocation [ a       geo:Instant ;
                          geo:lat  "56.114933895108166"^^xsd:double ;
                          geo:lon  "10.132573038028681"^^xsd:double
                        ] ;
        sao:hasType     ec:TransportationEvent ;
        tl:time         "2016-03-23T09:38:53.966Z"^^xsd:dateTime .
```

Figure 15: Noise status changes RDF event message

Listing 8: Detect Low noise events

```
insert into AarhusNoiseLevelStreamMedium select * from TrafficStream where
    (StreetVehicleCount * AverageSpeed > pollutionThresholdLow and
    StreetVehicleCount * AverageSpeed <= pollutionThresholdHigh)
```

Listing 9: Detect Medium noise events

```
insert into AarhusNoiseLevelStreamHigh select * from TrafficStream where (
    StreetVehicleCount * AverageSpeed > pollutionThresholdHigh)
```

Listing 10: Detect High noise events

A new event with a cetain level is generated when a status change is detected:

- Level = 0: when there is a transition from medium to low or from high to low level of noise.
- Level = 1: when there is a transition from low to medium or from high to medium level of noise.

- Level = 2: when there is a transition from low to high or from medium to high level of noise.

Listing 11 present the CEP statement used to detect the noise level change from low to medium but it can be changed to detect transitions between different states as presented above.

```
insert into AarhusNoiseLevelStreamLow2Medium select * from pattern [every
    AarhusNoiseLevelStreamLow -> AarhusNoiseLevelStreamMedium]
```

Listing 11: Status change noise events

4.1.3 Pollution event detection

The scope of the pollution level event detection node is to detect the changes of the status of the level of pollution in the city. To be more precise, is generates events when the number of cars on the street is higher or lower than some thresholds. In order to detect the level of pollution the node uses the observations generated by the traffic sensors deployed in Aarhus. It only needs one of it's features to detect the pollution level:

- vehicleCount: the number of cars passing through the sensing range of the sensor in the last 5 minutes.

Figure 16 depicts the main logic of the pollution level event detection node. The input in represented by one traffic stream from the city of Aarhus and the configuration parameters are:

- pollutionThresholdLow: the lower threhsold representing the minimum number of cars.
- pollutionThresholdHigh: the higher threhsold representing the maximum number of cars.

A Low level(level 0) pollution event is detected when the number of cars passing through the sensing area of the sensor is lower than *pollutionThresholdLow*. When the number of cars is in between the *pollutionThresholdLow* and *pollutionThresholdHigh* thresholds the node generates a Medium level(level 1) pollution event; and when the number of cars is above the upper threshold *pollutionThresholdHigh* the node generates a High level(level 2) pollution event.

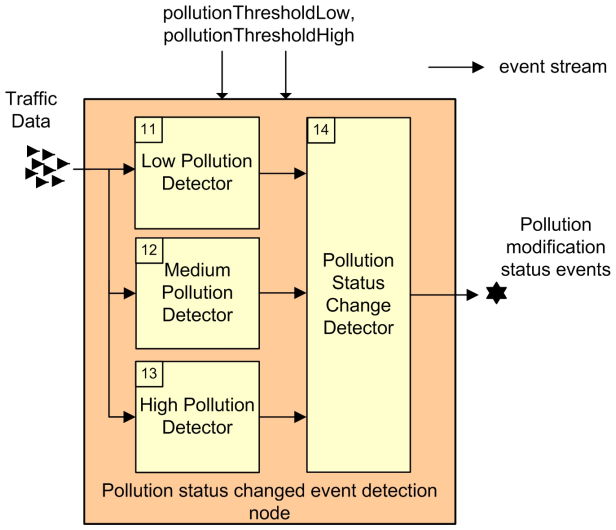

Figure 16: Pollution event detection node

The semantic Annotated data(Traffic Data) that is set as an input for the Pollution Level event detection node is the same as in the case of the Traffic Jam event detection node and is represented as a RDF message, with the format depicted in Figure 9.

Listing 12, 13 and 14 present the CEP statements used to detect the levels of pollution. There is a total of 3 statements that detect low, medium and high levels of pollution.

```
@prefix geo:   <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix sao:   <http://purl.oclc.org/NET/UNIS/sao/sao#> .
@prefix tl:    <http://purl.org/NET/c4dm/timeline.owl#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix prov:  <http://www.w3.org/ns/prov#> .
@prefix ec:    <http://purl.oclc.org/NET/UNIS/sao/ec#> .

sao:04e7aaf4-548f-409a-a7c9-49a2889bb5d7
        a               ec:AarhusPollution ;
        ec:hasSource    "SENSOR_74314cc6-d289-5292-b1b2-6f1c4f30008a" ;
        sao:hasLevel    "2"^^xsd:long ;
        sao:hasLocation  [ a        geo:Instant ;
                           geo:lat  "56.184723380602684"^^xsd:double ;
                           geo:lon  "10.21814102426572"^^xsd:double
                         ] ;
        sao:hasType     ec:TransportationEvent ;
        tl:time         "2016-03-14T14:59:54.108Z"^^xsd:dateTime .
```

Figure 17: Pollution status changes RDF event message

```
insert into AarhusPollutionStreamLow select * from TrafficStream where (
    StreetVehicleCount <= pollutionThresholdLow)
```

Listing 12: Detect Low pollution events

```
insert into AarhusPollutionStreamMedium select * from TrafficStream where (
    StreetVehicleCount > pollutionThresholdLow and StreetVehicleCount <=
    pollutionThresholdHigh)
```

Listing 13: Detect Medium pollution events

```
insert into AarhusPollutionStreamHigh select * from TrafficStream where (
    StreetVehicleCount > pollutionThresholdHigh)
```

Listing 14: Detect High pollution events

A new event with a cetain level is generated when a status change is detected:

- Level = 0: when there is a transition from medium to low or from high to low level of pollution.
- Level = 1: when there is a transition from low to medium or from high to medium level of pollution.
- Level = 2: when there is a transition from low to high or from medium to high level of pollution.

Listing 15 present the CEP statement used to detect the pollution level change from low to medium but it can be changed to detect transitions between different states as presented above.

```
insert into AarhusPollutionStreamLow2Medium select * from pattern [every
    AarhusPollutionStreamLow -> AarhusPollutionStreamMedium]
```

Listing 15: Status change pollution events

4.1.4 Event Detection node for Aarhus traffic prediction

The traffic jam event detection mechanism presented in subsection 4.1.1 is used to detect the traffic incidents at the occurrence time. In order to ensure the predictive behaviour of the system we have developed an prediction model which is used to forecast the traffic status on a particular area after T minutes.

During the design time of the ML model, we have used the historic traffic data from city of Aarhus. The traffic data stream is represented by an time series of observations having the features: number of cars and average speed. In order to do the prediction, the model uses the last n observations received from the data

stream. Before model training, a sliding window have been used to keep all the valid (without missing values) sequences on $n+1$ observations. At the end resulted a new dataset containing $n+1$ features. The first n features represents the last traffic observation and the last one represents the value to be predicted. As a result for this particular case T is equal with the traffic observation periodicity (5 minutes). We have also considered n as being 10. Figure 18 contains a graphic containing a representative subset of the training set, in order to have a intuition about how the data looks like. The red lines represent the first two features from the data data set and the black line represents the predicted value.

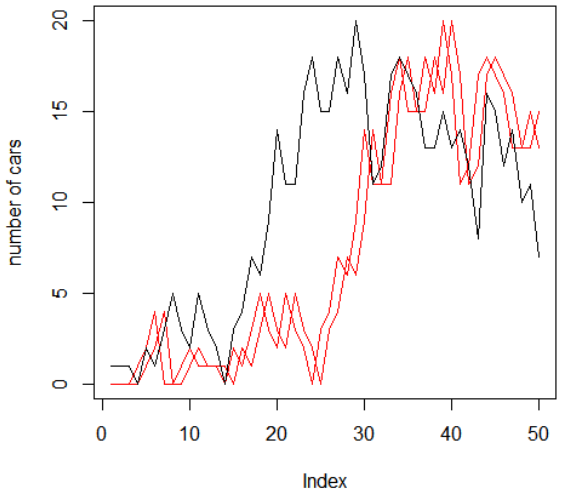

Figure 18: The training set of the prediction model (with red features 1 and 2 and with black the predicted value)

We have trained an neural network containing one input layer with 10 neurons and 1 output layer with one neuron (as presented in Figure 19).

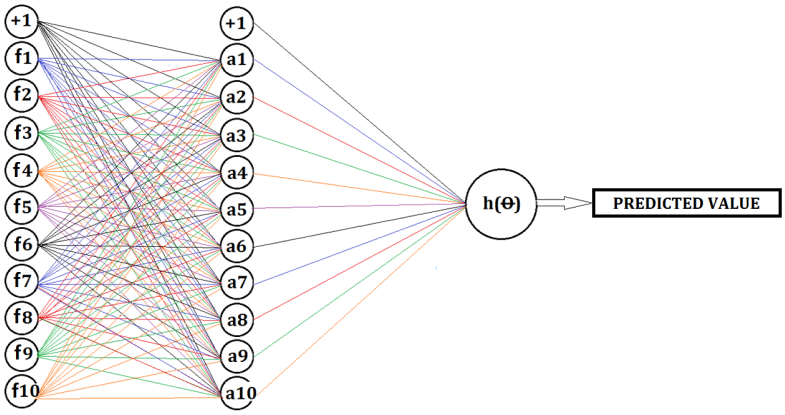

Figure 19: The topology of the neural network used for traffic prediction.

The obtained prediction model has been validated using a fresh test data set and the average error is 1.5 and the Pearson correlation parameter between actual and predicted values is 0.85. Figure 20 displays the predicted versus actual values.

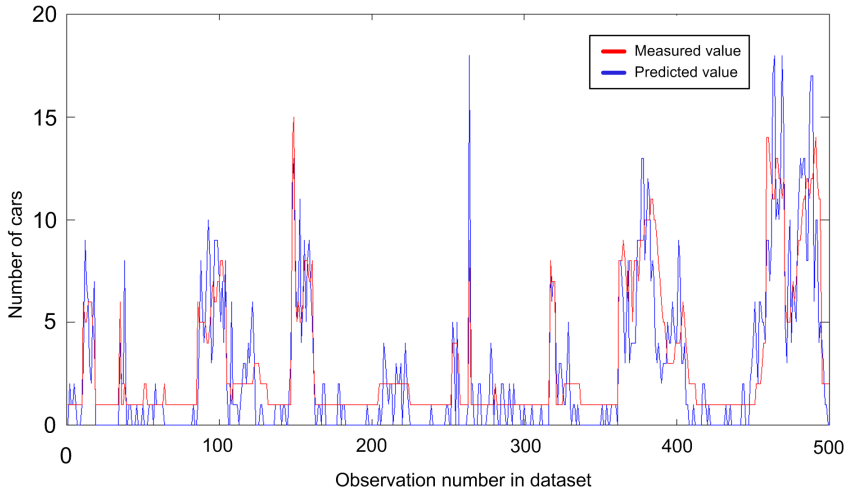

Figure 20: The performance of the neural network (predicted versus actual values).

4.1.5 Performance evaluation

The overall event throughput rate of the event detection component is determined by the number of nodes which are loaded and executed simultaneously in the instance of the component as well as their complexity. Yet, the first bottleneck we have encountered while doing the application stress tests was not driven by the number of loaded queries but by the communication side and the translation of the incoming data as events to be processed.

In order to provide the highest event input rate we have developed a test application which is able to replay the Aarhus traffic historic data as a high rate. Tests were performed on a commodity PC with an Intel i5-2410 2.30 GHz processor, 4GB of RAM, 1GB Ethernet adapter, running on a Windows 7, 64 bit operating system.

With 2,500 active nodes into the event detection component instance we have detected no delays, with a stable input rate varying around 15,000 events per second. In order to increase the stress on the input adapters we have increased the number of reporting sensors and reached a maximum relatively stable input rate of around 30,000 events per second. The test results for different event rates and number of nodes deployed in the event detection component are summarised in table 1.

Table 1: Event detection component stress tests results.

Event input rate [Events per second]	Active nodes	Communication Bottleneck	Event detection bottleneck
15,000	2,500	no	no
30,000	2,500	no	no
$> 30,000$	2,500	yes	no
15,000	5,000	no	no
15,000	$> 5,000$	no	yes
$> 30,000$	$> 5,000$	yes	yes

All the tests have been done using semantically annotated data. The event detection component can deliver the same performance when consuming aggregated data. As a result of that the amount of data which can be processed when the event detection component is used in combination with the data aggregation one depends on the variability of the data.

4.2 Event detection from social streams

We conducted our experiments on textual Twitter data collected from two geographically different locations: San Francisco Bay area (data collected by [48]) and our locally collected London data.

Our objective of the evaluation is to quantify the extent to which our results can extract city events from Twitter. We compare our approach with the state-of-the-art baselines [48, 55] and the results are reported in

terms of recall and precision.

4.2.1 Ground-truth Annotation Tool

To facilitate the ground-truth annotation of Tweets we have developed a tool with GUI. A view of this tool is represented in Fig. 21 [9].

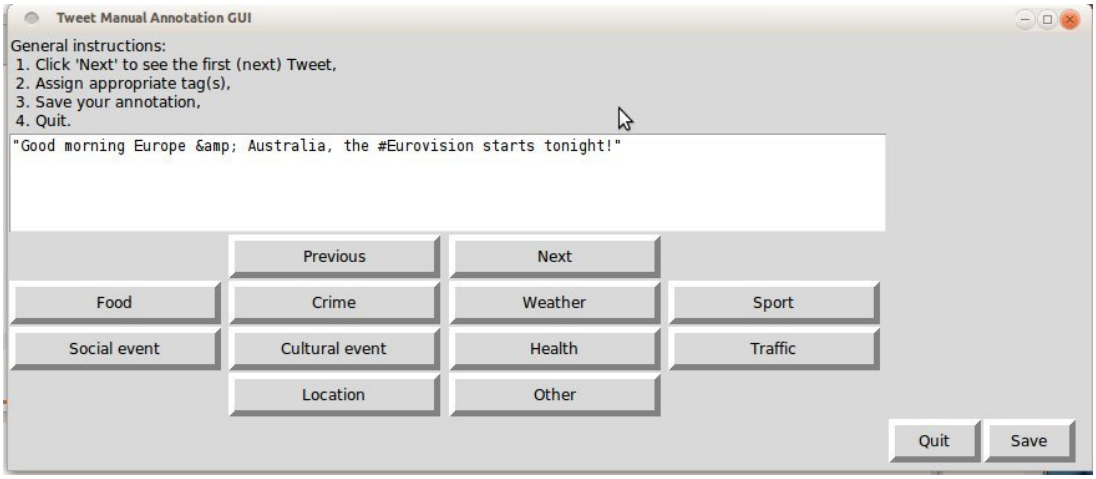

Figure 21: View of the Tweet annotation tool.

For the study, we have used this tool to annotate 3000 Tweets for constructing a training dataset for our model.

The annotation results also undergo a second investigation for ensuring their validity. We have asked a group of technical users who work on smart city research in our team to undertake the validation of the annotations.

4.2.2 Performance Evaluation

Tab. 2 shows a comparison between our proposed universal dictionaries efficiency and the two state-of-the-art approaches: B-1 [55] and B-2 [48]. We have used the dataset for this comparison as the two other baseline approaches had been evaluated using this data.

Table 2: Comparisons of our CRF dictionary tagging vs. the baseline B_1 and B_2 baseline methods.

	Other			Location			Events		
	B_1	B_2	Ours	B_1	B_2	Ours	B_1	B_2	Ours
Recall	0.93	0.98	0.98	0.75	0.85	0.92	0.24	0.88	0.85
Precision	0.93	0.98	0.97	0.82	0.93	0.95	0.11	0.86	0.84

It can be noted that our CRF+CNN dictionary chunking model performs equally well as the baseline models despite using generic city-independent dictionaries. This means we did not provide any domain-specific prior knowledge or controlled vocabulary to train our model and instead used a generic model (i.e.CNN model is trained on Wikipedia and not on a corpus of twitter text). However, our generic model performs as good as a model which is specifically trained for a controlled domain (San Francisco Bay area). This makes our model more flexible and adaptable to be used for different city events and also in other different domains.

Our proposed tagging approach is more flexible in the sense that it can be applied to Twitter data from other cities. This is due to the fact of benefiting from more generic set of dictionary terms which removes any geographical bias.

Tab. 3 shows the evaluation results of the proposed multi-view approach and Fig. 22 demonstrates the overall performance as a graph. The performance is reported in terms of one vs. all class accuracies. Fig. 22 also depicts the performance in comparison with the classification without multi-view learning.

[9] The annotation tool is available for download at: `http://goo.gl/UBTKQp`

Table 3: Numerical results of multi-view learning evaluation on Greater London data

1 vs. All Class Acc.	Crime	Cultural	Food	Social	Sport	Weather	Trans.
CRF + CNN word tagging annotation	0.53	0.36	0.35	0.16	0.52	0.8	0.67
multi-view Neural Network model annotation	0.6	0.37	0.48	0.21	0.54	0.8	0.69

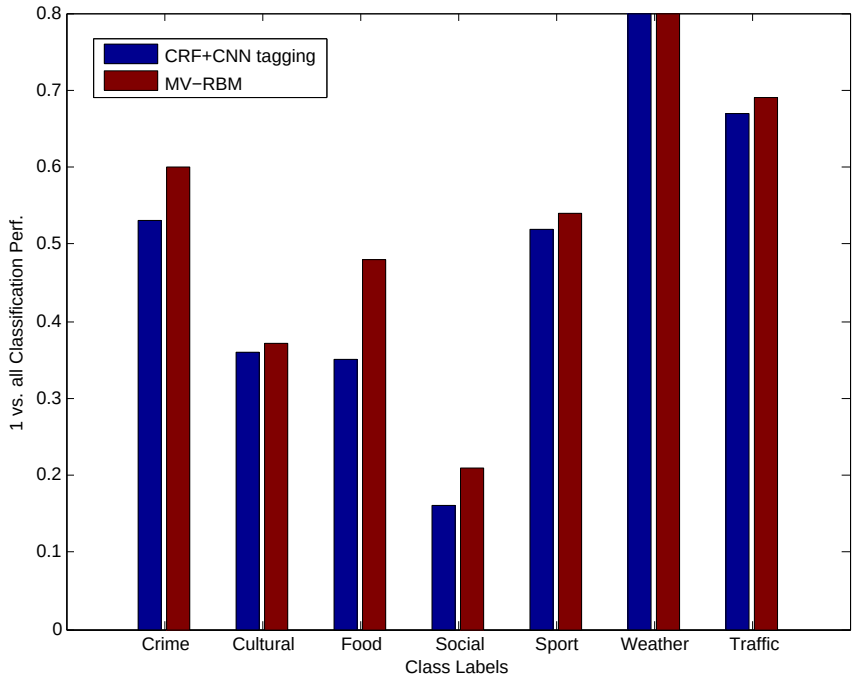

Figure 22: London data: 1 vs all classification performance result.

The enhanced model that uses the multi-view approach shows to better performance compared with the single view model that uses the CRF+CNN word tagging. The lower performance measure on classes such as *"Social", "Cultural","Food"* and *"Sport"* compared to other classes is mainly due to two reasons; incomplete class dictionaries and class dictionary term overlaps. Although extra cautions have been taken for constructing class-specific dictionaries, there yet exist terms and phrases which are included in more than one dictionary. This in practice makes the event extraction more challenging in such scenarios.

The effect of the dictionary problem can be also reduced by increasing the training data from the categories that provide less accurate results. However, we did not tend to bias our data by providing more training samples of these categories and reported the results based on fair number of annotated Tweets for our categories.

The multi-view training step can in fact provide more flexibility in expanding class-specific dictionaries. However, adding more terms will require the model to be trained with a larger size training data and will cause higher time and computational complexity and requires more manual annotation effort.

5 Conclusion

Providing enhanced services to citizens while cities are growing due to urbanisation, and while resources are limited demands for a more intelligent use of the existing resources. The cities have started to deploy sensor and actor devices in their environment, e.g. intelligent lighting, and observation and monitoring devices to collect traffic, air quality and water/waste data. At this moment there is a need to develop new techniques for processing and interpretation of streamed sensory and social media data in smart city environments.

The CityPulse framework exposes two modules to detect events happening in cities. The first module uses directly the sensory data sources from the city. The second one can be used to process the data from social media sources. In the case of CityPulse it is used for processing streams of tweets from Twitter.

The stream Event detection component provides the generic tools for processing the annotated as well as aggregated data streams to obtain events occurring into the city. This component is highly flexible in deploying new event detection mechanisms, since different smart city applications require different events to be detected from the same data sources. The component has been developed using the Esper engine [67].

The second module exposed by the Event detection component is used to process the Social Media streams. The module analyses and annotates large-scale real-time Twitter data streams.

To demonstrate the event detection component ease of use in integrating the raw city data and in detecting various problems, we have applied it for processing the traffic data from the city of Aarhus. In total the data of 449 traffic sensors and 13 parking garages data streams have been piped into a running instance of our CityPulse framework. Then two event detection nodes have been implemented for detecting traffic jams and the moments when the occupancy level into the parking garages changes significantly. In addition to these, two more event detection nodes have been developed for estimating the noise and pollution level.

Within the CityPulse framework, the real-time extracted city events are then used by *Real-time Adaptive Urban Reasoning* component to obtain a more comprehensive interpretation of the city events.

The social media extracted events are utilised to complement the sensor stream information extraction and allow obtaining a more detailed interpretation of the city events.

References

[1] Neal Leavitt. Complex-Event Processing Poised for Growth. *Computer*, 42(4):17–20, April 2009.

[2] David C. Luckham. Rapide: A language and toolset for simulation of distributed systems by partial orderings of events. In *Princeton University*, 1996.

[3] Opher Etzion and Peter Niblett. *Event Processing in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2010.

[4] B.M. Michelson. Event-Driven Architecture Overview - Event-Driven SOA Is Just Part of the EDA Story. Technical report, Patricia Seybold Group/Business-Driven Architecture, 2006.

[5] K Vidačković, T. Renner, and S Rex. Market Overview Real-Time Monitoring Software, Review of Event Processing Tools. Technical report, Fraunhofer IAO, 2010.

[6] Mike Gualtieri and John R. Rymer. The Forrester Wave: Complex Event Processing Platforms, Q3 2009. Technical report, Forrester Research, Inc., Aug. 2009.

[7] K. Chandy and W. Schulte. *Event Processing: Designing IT Systems for Agile Companies*. McGraw-Hill, Inc., New York USA, 1 edition, 2010.

[8] Bartosz Balis, Bartosz Kowalewski, and Marian Bubak. Real-time Grid Monitoring based on Complex Event Processing. *Future Generation Computer Systems*, 27(8):1103–1112, 2011.

[9] M.J.A.G. Izaguirre, A. Lobov, and J.L.M. Lastra. OPC-UA and DPWS Interoperability for Factory Floor Monitoring using Complex Event Processing. In *9th IEEE International Conference on Industrial Informatics (INDIN)*, pages 205 –211, July 2011.

[10] S. Karadgi, D. Metz, M. Grauer, and W. Schafer. An Event Driven Software Framework for Enabling Enterprise Integration and Control of Enterprise Processes. In *10th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 24–30, Dec. 2010.

[11] K. Walzer, J. Rode, D. Wunsch, and M. Groch. Event-driven Manufacturing: Unified Management of Primitive and Complex Events for Manufacturing Monitoring and Control. In *IEEE International Workshop on Factory Communication Systems*, pages 383–391, May 2008.

[12] P. Rosales, Kyuhyup Oh, Kyuri Kim, and Jae-Yoon Jung. Leveraging Business Process Management Through Complex Event Processing for RFID and Sensor Networks. In *40th International Conference on Computers and Industrial Engineering (CIE)*, pages 1–6, July 2010.

[13] Yintai Ao, Wei He, Xuejian Xiao, and Engwah Lee. A Business Process Management Approach for RFID Enabled Supply Chain Management. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–7, Sept. 2010.

[14] C.M. Wu, R. S Chang, and C. C. Chen. A Complex Event Processing Method based on Pre-Grouping for Radio Frequency Identification Data Streams. In *IET International Conference on Frontier Computing. Theory, Technologies and Applications*, pages 133–138, 2010.

[15] Yanming Nie, Zhanhuai Li, and Qun Chen. Complex Event Processing over Unreliable RFID Data Streams. In *Proceedings of the 13th Asia-Pacific Web Conference on Web Technologies and Applications*, APWeb'11, pages 278–289, Berlin, Heidelberg, 2011. Springer-Verlag.

[16] J. Dunkel. On Complex Event Processing for Sensor Networks. In *International Symposium on Autonomous Decentralized Systems*, pages 1–6, March 2009.

[17] Jürgen Dunkel, Alberto Fernández, Rubén Ortiz, and Sascha Ossowski. Event-Driven Architecture for Decision Support in Traffic Management Systems. *Expert Systems with Applications*, 38(6):6530–6539, June 2011.

[18] Fernando Terroso-Sáenz, Mercedes Valdés-Vela, Francisco Campuzano, Juan A. Botia, and Antonio F. Skarmeta-Gómez. A Complex Event Processing Aapproach to Perceive the Vehicular Context. *Information Fusion*, in press, 2012.

[19] F. Terroso-Saenz, M. Valdes-Vela, C. Sotomayor-Martinez, R. Toledo-Moreo, and A.F. Gomez-Skarmeta. A Cooperative Approach to Traffic Congestion Detection With Complex Event Processing and VANET. *IEEE Transactions on Intelligent Transportation Systems*, 13(2):914–929, June 2012.

[20] Tom M. Mitchell. The need for biases in learning generalizations. Technical report, Rutgers University, New Brunswick, NJ, 1980.

[21] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[22] S.S. Haykin. *Neural Networks and Learning Machines*. Number v. 10 in Neural networks and learning machines. Prentice Hall, 2009.

[23] B. Schölkopf and A.J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive computation and machine learning. MIT Press, 2002.

[24] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive computation and machine learning. MIT Press, 2009.

[25] Giovanni Seni and John Elder. *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*. Morgan and Claypool Publishers, 2010.

[26] Nikunj C. Oza. Ensemble data mining methods. In John Wang, editor, *Encyclopedia of Data Warehousing and Mining*, volume 1, pages 448–453. Idea Group Reference, 2006.

[27] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC, 1st edition, 2012.

[28] C. Zhang and Y. Ma. *Ensemble Machine Learning: Methods and Applications*. Springer, 2012.

[29] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[30] Oded Maimon and Lior Rokach. *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[31] Benjamin M. Marlin. *Missing Data Problems in Machine Learning*. PhD thesis, University of Toronto, 2008.

[32] M. Randolph-Gips. A new neural network to process missing data without imputation. In *Machine Learning and Applications, 2008. ICMLA '08. Seventh International Conference on*, pages 756–762, Dec 2008.

[33] Yoshua Bengio and Francois Gingras. Recurrent neural networks for missing or asynchronous data. In David S. Touretzky, Michael Mozer, and Michael E. Hasselmo, editors, *NIPS*, pages 395–401. MIT Press, 1995.

[34] I. Guyon. *Feature Extraction: Foundations and Applications*. Studies in Fuzziness and Soft Computing. Springer, 2006.

[35] Huan Liu and Hiroshi Motoda. *Computational Methods of Feature Selection (Chapman & Hall/Crc Data Mining and Knowledge Discovery Series)*. Chapman & Hall/CRC, 2007.

[36] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, March 2003.

[37] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. Also published as a book. Now Publishers, 2009.

[38] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[39] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In P. Bartlett, F.c.n. Pereira, C.j.c. Burges, L. Bottou, and K.q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1106–1114. 2012.

[41] Richard Socher and Eric H. Huang and Jeffrey Pennington and Andrew Y. Ng and Christopher D. Manning. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In *Advances in Neural Information Processing Systems 24*. 2011.

[42] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

[43] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005.

[44] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[45] Ruslan Salakhutdinov and Geoffrey Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 11, 2007.

[46] Jennifer Bélissent. Getting clever about smart cities: new opportunities require new business models. 2010.

[47] Jennifer Bélissent and Frederic Giron. Service providers accelerate smart city projects. July 2013.

[48] Pramod Anantharam, Payam Barnaghi, Krishnaprasad Thirunarayan, and Amit P. Sheth. Extracting city traffic events from social streams. In *ACM Transactions on Intelligent Systems and Technology*, volume -, New York, NY, USA, 2015. ACM.

[49] Dunja Mladenić and Alexandra Moraru. Complex event processing and data mining for smart cities. 2012.

[50] Mingrong Liu, Yicen Liu, Liang Xiang, Xing Chen, and Qing Yang. Extracting key entities and significant events from online daily news. In *Intelligent Data Engineering and Automated Learning - IDEAL 2008, 9th International Conference, Daejeon, South Korea, November 2-5, 2008, Proceedings*, pages 201–209, 2008.

[51] Masayuki Okamoto and Masaaki Kikuchi. Discovering volatile events in your neighborhood: Local-area topic extraction from blog entries. In Gary Geunbae Lee, Dawei Song, Chin-Yew Lin, Akiko N. Aizawa, Kazuko Kuriyama, Masaharu Yoshioka, and Tetsuya Sakai, editors, *AIRS*, volume 5839 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 2009.

[52] Hristo Tanev, Jakub Piskorski, and Martin Atkinson. Real-time news event extraction for global crisis monitoring. In Epaminondas Kapetanios, Vijayan Sugumaran, and Myra Spiliopoulou, editors, *NLDB*, volume 5039 of *Lecture Notes in Computer Science*, pages 207–218. Springer, 2008.

[53] Ralph Grishman, Silja Huttunen, and Roman Yangarber. Real-time event extraction for infectious disease outbreaks. In *Proceedings of the Second International Conference on Human Language Technology Research*, HLT '02, pages 366–369, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[54] Hila Becker, Mor Naaman, and Luis Gravano. Beyond trending topics: Real-world event identification on twitter. In Lada A. Adamic, Ricardo A. Baeza-Yates, and Scott Counts, editors, *Proceedings of the Fifth International Conference on Weblogs and Social Media, Barcelona, Catalonia, Spain, July 17-21, 2011*. The AAAI Press, 2011.

[55] Alan Ritter, Mausam, Oren Etzioni, and Sam Clark. Open domain event extraction from twitter. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 1104–1112, New York, NY, USA, 2012. ACM.

[56] Xiaofeng Wang, Matthew S Gerber, and Donald E Brown. Automatic crime prediction using events extracted from twitter posts. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 231–238. Springer, 2012.

[57] Lluís Màrquez, Xavier Carreras, Kenneth C. Litkowski, and Suzanne Stevenson. Semantic role labeling: An introduction to the special issue. *Comput. Linguist.*, 34(2):145–159, June 2008.

[58] Vasileios Lampos and Nello Cristianini. Nowcasting events from the social web with statistical learning. *ACM Trans. Intell. Syst. Technol.*, 3(4):72:1–72:22, September 2012.

[59] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November 2011.

[60] Cicero dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78. Dublin City University and Association for Computational Linguistics, 2014.

[61] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of theTwenty-eight International Conference on Machine Learning (ICML'11)*, volume 27, pages 97–110, June 2011.

[62] Duyu Tang, Furu Wei, Bing Qin, Ting Liu, and Ming Zhou. Coooolll: A deep learning system for twitter sentiment classification. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 208–212. Association for Computational Linguistics, 2014.

[63] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1), 2009.

[64] ***. Time Series Analysis and Forecasting with Weka. *Pentaho*, Retrieved 2013. accessed Aug. 2013.

[65] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[66] Alias-i. Lingpipe 4.1.0., 2008.

[67] ***. Esper Complex Event Processing Engine. *EsperTech*, 2012. accessed October 2015.