PlanetData

Network of Excellence

FP7 – 257641

D1.6 Composed event detection in multiple data sources

Coordinator: Carolina Fortuna
With contributions from: Alexandra Moraru, Calin Railean, Marko Grobelnik

1st Quality reviewer: Giorgos Flouris
2nd Quality reviewer: Jean-Paul Calbimonte

Deliverable nature:	R
Dissemination level: (Confidentiality)	PU
Contractual delivery date:	M36
Actual delivery date:	27.09.2013
Version:	0.4
Total number of pages:	27
Keywords:	Non-structured data, social media, data modalities, complex event processing, stream, stream processing.

Abstract

Complex Event Processing (CEP) is emerging as a new paradigm for continuous processing of streaming data in order to detect relevant information and provide support for timely reactions. The main role of a CEP engine is to detect the occurrence of event patterns on the incoming stream of data. This deliverable reports on the state of the art in the area of CEP identifying also existing research challenges and analyses two smart city use cases. For the two use cases, we collected and pre-processed over 1.5 years of data from several smart city related data sources. The first challenge was to enable an efficient browsing and visualization functionality on this data while the second was to identify possible correlations between the data sources. The indirect implication of the tasks is that they enable us to identify ways to improve the existing state of the art and tools from the area of CEP. Both tasks are pre-requisites for enabling stream mining of events.

[End of abstract]

Executive summary

Complex Event Processing (CEP) is emerging as a new paradigm for continuous processing of streaming data in order to detect relevant information and provide support for timely reactions. The key concept in the area is the *event* which can represent anything that happens or is observed as happening. A common characteristic of event processing applications is to continuously receive events from different event sources. The main role of a CEP engine is to detect the occurrence of event patterns on the incoming streams of data.

In this deliverable, we first provide an overview of the state of the art in the field of CEP that includes an overview of the available languages and the applications that naturally require such technology. Then, two categories of event processing languages are identified: stream-oriented and rule-oriented. The stream-oriented languages specify operations for processing the input streams in order to obtain some other output streams. The rule-oriented languages specify rules for processing streams, clearly separating the triggering conditions and the actions to be taken when the conditions are met. We then continue with an overview of three application domains where CEP systems can provide an obvious advantage (RFID applications, sensor networks and financial systems) and discuss the features required from the systems with respect to these applications.

Next, a critical overview is provided and possible improvements of CEP systems are analysed. One of the significant findings emerging from this study is that few of the research approaches analysed has given a solution for the integration of machine learning or data mining algorithms into CEP systems for direct support in the definition of event patterns. Another critical aspect is that the systems built for different applications make certain assumptions (especially the ones using simulated data) such as the quality of data or the reliability of the communication channels that can lead to unpredictable behaviour when such systems are deployed in real world environments.

The last part of the deliverable focuses on using CEP for a smart city scenario. Traffic, weather, public events and social network data have been collected for over 1.5 years from publicly available data sources from London. Adapters for two main CEP tools –StreamInsight and Esper – were built for this data. An application using the StreamInsight engine for visualizing in real time the moving patterns of the public bikes is then described and was built for data exploration purposes. Then, using the Esper engine, correlations between tweets and public events were investigated. The plan is to continue developing more complex scenarios for these applications in which the integration of machine learning techniques and CEP systems will be investigated.

Document Information

IST Project Number	FP7 - 257641	**Acronym**	PlanetData
Full Title	PlanetData		
Project URL	http://www.planet-data.eu/		
Document URL			
EU Project Officer	Leonhard Maqua		

Deliverable	**Number**	D1.6	**Title**	Composed event detection in multiple data sources
Work Package	**Number**	WP1	**Title**	Data Streams and Dynamicity

Date of Delivery	**Contractual**	M36	**Actual**	M36
Status	version 1.0		final □	
Nature	prototype □ report X dissemination □			
Dissemination level	public X consortium □			

Authors (Partner)	Alexandra Moraru (IJS), Calin Railean (IJS/TUC-N), Marko Grobelnik (IJS), Carolina Fortuna (IJS)			
Responsible Author	**Name**	Carolina Fortuna	**E-mail**	carolina.fortuna@ijs.si
	Partner	IJS	**Phone**	+ 386 1 477 3144

Abstract (for dissemination)	
Keywords	

Version Log			
Issue Date	**Rev. No.**	**Author**	**Change**
2013-02-06	0.0	Carolina Fortuna	Initial draft of ToC
2013-02-07	0.1	Alexandra Moraru	Revised ToC
2013-08-15	0.2	Carolina Fortuna	Initial content
2013-08-30	0.3	Alexandra Moraru, Calin Railean	Additional content
2013-09-02	0.4	Carolina Fortuna	Small corrections
2013-09-12	0.5	Alexandra Moraru	Additional content
2013-09-26	0.6	Alexandra Moraru	Addressed reviewer comments
2014-05-22	0.7	Alexandra Moraru	Additional content on CEP for large scale data
2014-05-26	1.0	Carolina Fortuna	Review for spelling.

Table of Contents

List of figures

List of tables

1 Introduction

The avalanche of data which information systems have been facing over the last years influenced their evolution and characteristics. When dealing with large scale data, two main directions can be identified for the developed information systems: (1) continuous stream processing systems and (2) large-scale data management systems. The first type of systems are the focus of this deliverable, while the second type refer to large-scale data management tools, which are typically deployed on local-area clusters or cloud infrastructures, and more details can be found in Deliverable 5.2 [1].

Continuous, on-time processing of incoming data streams impose particular requirements [2], which traditional Database Management Systems (DBMS) are not able to fulfil. Consequently, new tools that are able to process multiple (streaming) data sources in a timely fashion have been developed. Grouped under the domain of *event processing* (or, according to [3] information flow processing domain), two main types of such systems have emerged: Data Stream Management Systems (DSMS) and Complex Event Processing (CEP) systems.

The term *event processing* refers to a broad research area, similar to terms like *machine learning* or *data mining.* In [4], the term of *event processing* is coined to "any form of computing that performs operations on events", where an event can be anything that has happened or is perceived as having happened. Examples of application areas include: environmental monitoring using sensor networks [5][6][7], RFID-enabled monitoring in logistics or building management [8][9][10], trading in the financial markets [11][12], etc.

With a relatively short research history [13], the terminology used in the area of event processing is yet to be harmonized and formalized, in order to satisfy different communities and vendors that are involved. We summarize below the basic concepts used throughout this deliverable, adapted from [3][4][14].

The key concept in the area is the **event** which can represent anything that happens or is observed as happening (e.g. a mouse click, a sensor reading, water level increase, a river flood, spring coming, etc.). A common characteristic of event processing applications is to continuously receive events from different **event sources** (e.g. sensors, software modules, blogs, etc.). The central module processing the events, called the CEP **engine**, detects **event patterns** from the incoming data streams (i.e. streams of simple events) and outputs the detected or predicted complex events which can then be further used by other **event consumers**, or it can even return as an input to the CEP engine itself forming a feedback loop. An illustration of these concepts and the links between them is provided in Figure 1. A general definition of a **complex event** is given as an abstraction of other events, called its members. There is no exact limit at which an event is considered complex, being depended on the application domain. For instance, the event of an increase in water level can be considered as a simple event by applications for flood prediction or as a complex event by applications processing raw sensor data.

The event pattern's role is to specify how the incoming events should be processed in order to extract relevant information. The language used to define these patterns should have the ability of specifying complex relationships among events flowing into the CEP engine. Different types of languages exist and they can be classified by their style in two main categories: stream-oriented and rule-oriented [4]. A more detailed characterization of these languages is given in Section 2.1.

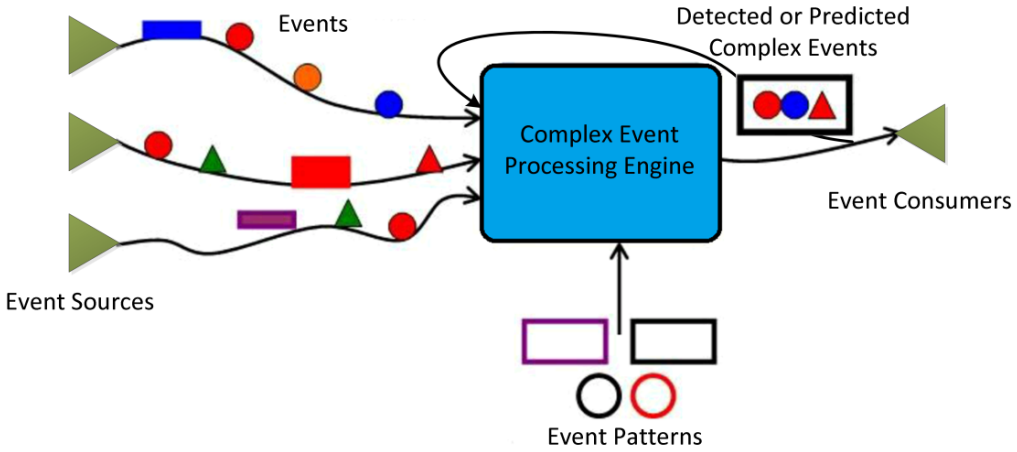

Figure 1. High-Level overview of a CEP (adapted from thecepblog.com).

1.1 Requirements

Event processing is emerging as a solid area of research serving application domains where **effective responses** are needed as a result of occurring events. Considering the reviewed literature [1][3][4][13][14], we would like to emphasize on two general requirements, that we consider representative for the area of CEP:

- **Timely processing** – a general understanding of this requirement is to process the incoming events as they come in (or on-line processing) and not at a later time in a batch. However, in some of the application domains with very tight requirements, such as trading market [15], where event latencies of 1 second are not acceptable, the meaning of this requirement is reduced to the response time of the system and referred to as real-time processing.
- Handling **high volume of events** – the volume of events is given by the number of event sources and the event throughput, which is the rate of incoming events that must be handled within a specified time interval (i.e. below the specified latency).

These two main requirements must be satisfied for dynamic scenarios, which means that the systems must provide flexibility for easy (re)configuration regarding (1) the number and types of event sources, (2) the representation of events, (3) the number and types of event consumers to which the results are transmitted.

2 State of the Art in Complex Event Processing

Several systems have been developed in the last years, as *complex event processing* or *event stream processing* became buzz-words in the context of "big data". In an extensive survey of existing systems for processing flows of information [3], an abstract framework was proposed for describing the main functionalities of a CEP engine[1]. We adopt this framework as state of the art for the CEP domain and describe it briefly.

The main components of the functional architecture are illustrated in Figure 2 and can be summarized as follows. When a new event is detected by an event observer, it is sent to the CEP engine by the *Receiver* component, which acts as a wrapper for the incoming stream of events. Depending on the application, the Receiver may (1) act as a demultiplexer by receiving events from multiple sources and sending them further one by one, (2) be connected to an internal clock for periodic processing (e.g. events may be processed every hour and not immediately when they are generated), (3) perform some pre-processing of the events.

In order to distinguishing between the two main phases of event processing (1) the *detection phase* and (2) the *production* (result) *phase*, the CEP engine has been divided in two main sub-components: the *Decider* and the *Producer*. The role of these two components is to process the incoming events according to a set of *Rules.* From a logical point of view, a processing rule is defined by two components ($C \rightarrow A$): a condition C and an action A. The condition specifies the event pattern (e.g. sensor reading exceeding some threshold) that is continuously checked by the Decider on the arrival of new events. When an event pattern is detected, the corresponding action is sent to the Producer, which generates the result (e.g. an alarm, creation of a new complex event) for the event sequence that triggered the specific rule. The result produced is sent to event consumers (through the Forwarder), or it can also be sent internally, to be processed again (recursive processing). The *Knowledge Base* component is an optional component that can store the static information needed for event processing.

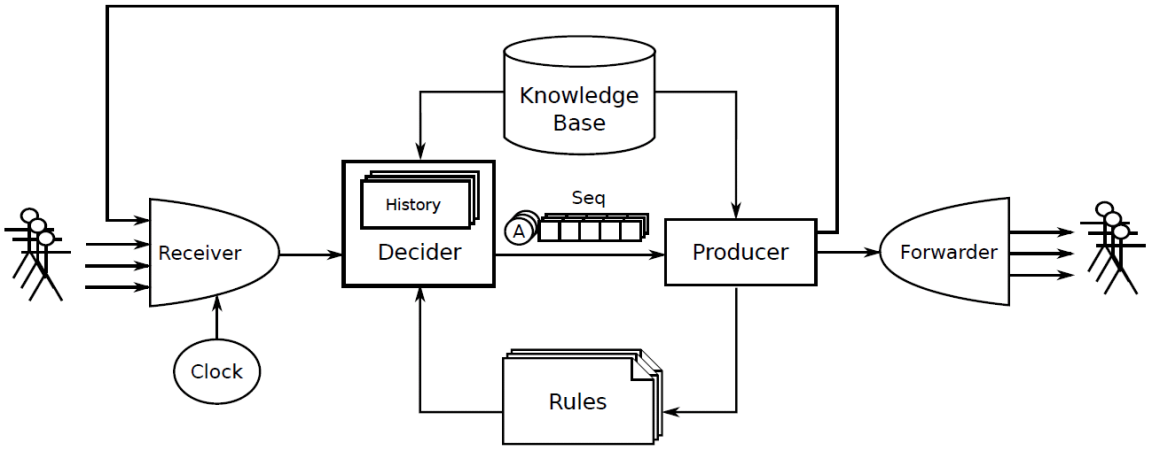

Figure 2. The functional architecture of a CEP system [3]

2.1 Event Processing Languages

Two categories of event processing languages can be distinguished: stream-oriented (or transforming languages) and rule-oriented (or detecting languages). The stream-oriented languages specify operations for processing the input streams (such as filtering, joining or aggregating) in order to obtain some other output streams. The rule-oriented languages specify rules for processing streams, clearly separating the triggering conditions and the actions to be taken when the conditions are met (the conditions represent event patterns). Stream-oriented languages are typically used within DSMS, while rule-oriented languages are used in CEP systems.

[1] The authors prefer to use the term Information Flow Processing (IFP) system for setting a new level of abstraction that would help in clearing the misunderstandings which have been created between different communities around the term of complex event processing. For consistency with the rest of this document we stick to the CEP term.

2.1.1 Stream-Oriented Languages

For the stream-oriented languages, the Continuous Query Language (CQL) [16] sets the ground in terms of abstract semantics and for the most representative operators used in DSMS. The CQL language has been developed in the STREAM[2] project and is currently adopted by commercial systems. (e.g. Oracle). Built as an extension of a standard relational database language, namely Structured Query Language (SQL), CQL is used for registering continuous queries over streams. The general idea is to convert data streams, which contain event tuples, into relations on which regular SQL queries can be evaluated.

Considering an ordered time domain *T*, we can define a stream as follows:

A *stream S* is a (possibly infinite) bag (multiset) of *elements* $\langle s, t \rangle$, where *s* is a tuple belonging to the schema of *S* and $t \in T$ is the timestamp of the element.

Furthermore:

A relation *R* is a mapping from *T* to a finite but unbounded bag of tuples belonging to the schema of *R*.

The CQL language is built around three classes of operators over *streams* and *relations* (see Figure 3):

- stream-to-relation operators , producing a relation *R* from a stream *S*
- relation-to-relation operators, producing a relation *R* from one or more other relations $R_1, R_2, \ldots R_n$
- relation-to-stream operators, producing a stream *S* from a relation *R*,

where each stream and relation has a fixed schema consisting of a set of named attributes and, in addition, a stream will always have a *timestamp* attribute. Stream-to-stream operators have to be composed from operators of the three classes above. The scope of CQL language was to use the existing knowledge from the relational model, giving a more intuitive way of interacting with streams.

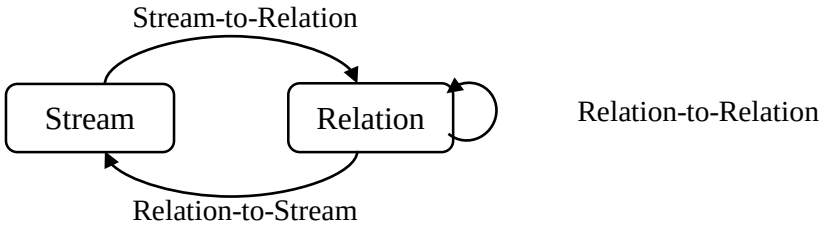

Figure 3. Main classes of operators for stream based languages [16]

Stream-to-Relation Operators

The main class of stream-to-relation operators is the *sliding window*. There are three types of sliding widow operators:

- *time based* windows, which capture the last segment of an ordered stream for a given time interval. Example: `S [Range 40 Seconds]` returns a relation *R* with all the elements of *S* from the last 40 seconds.
- *tuple based* windows, which capture the last N elements of a stream, where N is given as a parameter. Example: `S [Rows 10]` returns a relation *R* with the last 10 elements of *S*.
- *partitioned* windows, which work similarly to a tuple based window, with an additional constraint on the stream attributes. Example: `S [Partition By` A_1`, …,` A_k `Rows N]` partitions *S* into different sub-streams based on equality of the attributes (similar to `Group By` from SQL) , computes a tuple based sliding window of size N on each sub-stream and then takes the union of these windows to produce the output relation.

Relation-to-Relation Operators

The relation-to-relation operators are adopted from the traditional relation operators of SQL (e.g.: *Select, Project, Join, Union, Intersect*, etc.)

Relation-to-Stream Operators

- Insert Stream – *Istream (R)*, applied to relation R contains a stream element *(s, t)* such that each *s* is in relation *R* at time *t*, but was not in *R* at time *t* -1.

[2] http://infolab.stanford.edu/stream/

- Delete Stream – *Dstream(R)*, applied to relation R contains a stream element *(s, t)* such that each *s* was in relation R at time *t* -1, but is not in R at time *t*.
- Relation Stream – *Rstream(R)*, applied to relation R contains a stream element *(s, t)* such that each *s* is in relation R at time *t*.

2.1.2 Rule-Oriented Languages

Rules, as a method of event processing, have been introduced first in the area of Active DBMSs. The HiPAC project proposed the Event-Condition-Action (ECA) rules as a formalism for defining the actions that should be executed when specific conditions are met [17]. An ECA type of rule has the following abstract syntax:

```
ON event IF condition DO Action
```

and the semantic meaning that the *action* is executed if the *condition* is satisfied when the *event* occurs. Therefore, a rule is evaluated only when triggered by a specific event. This event can be primitive (database operations, temporal events or external notifications) or a combination of primitive events using logic operators. After the rule is triggered, the condition is evaluated as a collection of queries, which may include attributes of the events. If the queries return non-empty results the action is executed. The steps described above can be executed in one or more database transactions, depending on the setting of coupling parameters (event-condition coupling and condition-action coupling). An illustrative example of an ECA rule is [17]:

```
Event:          update Xerox Corp. stock price
Condition:      where new price = 50
Action:         send request to buy 500 shares for client A
```

Complex event patterns in active DBMS have captured attention in the context of cause-effect problems for domains such as stock trading, trend computation [18]. One of the languages developed for event specification, called Snoop [19], has also introduced operators for constructing complex events. A classification of events by their type is clearly defined into **primitive events**, which are pre-defined in the system and **composite or complex events**, which are formed by applying a set of operators to primitive and composite events (see Figure 4). Further classification of primitive events is done into three categories: database events, temporal events and explicit events. Database events are database operations such as transactions, insert, read, update and delete operations. Temporal events can be absolute, which are discrete points in time (e.g. 12 p.m.) or relative, which are defined by a reference point and an explicit offset (e.g. 3 seconds after an event). Explicit events come from external application programs and their parameters must be registered in the system (e.g. there is a relation describing such events).

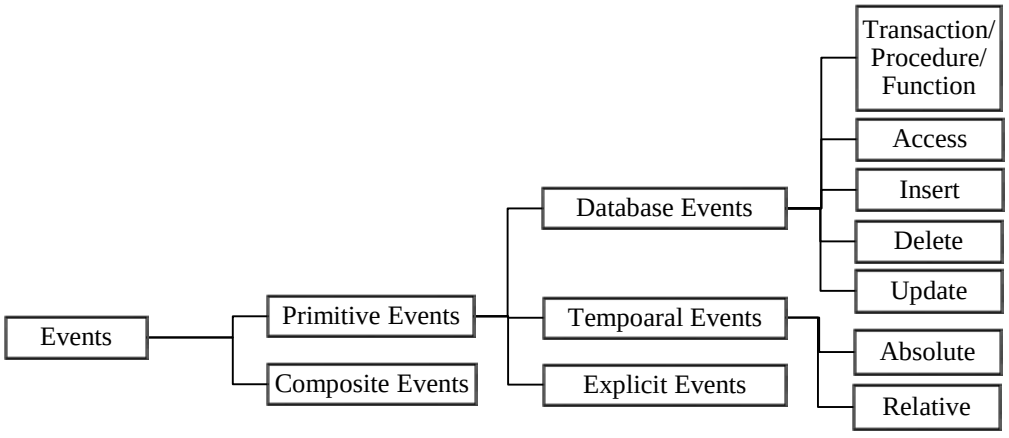

Figure 4. Event Classification in Active DBMS [19]

Composite events are created using the following operators: logic operators, any, sequence, periodic event and aperiodic event, as listed in Table 1. These are basic operators which have been extended by other

languages for representing composed events. The two most used logic operators are disjunction and conjunction. Disjunction of two events is satisfied when at least one event is detected; similarly, conjunction of two events is satisfied when both events are detected, irrespective of their order. The negation operator is a special case, which is described in the discussion section. The *Any* operator identifies occurrence of m distinct events out of the n events specified. The sequence SEQ(;) of two events is detected when the second event occurs after the first event had occurred. The aperiodic operator expresses the occurrences of an aperiodic event in a time interval, with two flavours, cumulative and non-cumulative. The periodic operator represents temporal events that occur periodically. More details can be found in Table 1.

Table 1. Event Operators used for defining Composed Events

Operator	**Representation**	**Meaning**
Disjunction (OR)	$E_1 \vee E_2$	E_1 or E_2 occurs
Conjunction (AND)	$E_1 \wedge E_2$	Both E_1 and E_2 occur, irrespective of their order
Sequence (SEQ ;)	E_1 ; E_2	E_2 occurs after E_1 had occurred
Any	ANY(m, $E_1, E_2, \ldots, E_n$), where $m \leq n$	Denotes the occurrence of m events out of the n events specified.
Aperiodic	A(E_1, E_2, E_3) (non-cumulative)	Each occurrence of E_2 during the closed interval defined by the occurrence of E_1 and E_3
	A*(E_1, E_2, E_3) (cumulative)	Only one occurrence of E_2, after E_3 occurs and accumulates the parameters for each occurrence of E_2.
Periodic	P(E1, t[:parameters], E3)	Collects the specified parameters every *t* time units, starting when E_1 occurs and ending after E_3. t[:parameters] is a constant time increment with optional parameter list
	P*(E1, t[:parameters], E3)	Collects the specified parameters every *t* time units, starting when E_1 and making them available after E_3 occurs.

Further research work was done for handling the problem of occurrence vs. detection of composite events, debating the problem of how to define the timestamps for the composite events [20]. The solution given by the authors is to use interval-based semantics in order to overcome the problems that are caused by the detection based semantics.

2.1.3 Discussion

In the sections above we have described the important characteristics of the two main types of languages used in event processing systems, namely stream oriented and rule oriented languages. Several systems have been developed that implemented these languages and a detailed list of the implemented operators is presented in [3]. However, there is no language that has proven general and complete enough to satisfy all the needs which can be encountered in CEP applications. Obviously, the more operators are available, the more expressivity is achieved. Nevertheless, for simple applications there might be more need for efficiency than for expressivity, and vice-versa, complex applications need more expressivity.

Another observation regarding the operators presented is that their separation between the two types of languages tends to be rather conceptual while in practice many of the CEP languages proposed and developed later, implement operators from both types of languages besides extending them with new ones [8][9][10][21][22].

Non-occurrence of Events

The problem of representing non-occurrence of events cannot be easily expressed in many existing systems. In active DMBS the non-occurrence (negation operator) of an event has been first defined for special cases, with well-defined intervals (i.e. Not(E_1, E_2, E_3)) occurs when E_2 occurs or when E_3 occurs if E_2 has not occurred in the closed interval defined by E_1 and E_3) [18]. However, more expressive operators may be required, as it is

illustrated in [20], where the language proposed for the complex event detection and response (CEDR) systems, supports three negation operators applied within a scope that can be time based or sequence based.

Exception Handling

Another difficult aspect is that of handling exceptions, as it was not possible to ignore specific instances from a pre-defined set (e.g. every working day would need to ignore weekend days) [19].

2.2 CEP Applications

In this section we are going to (1) analyse three domains of application for CEP systems: RFID applications, sensor networks and financial systems and (2) discuss several characterises of CEP systems with respect to the application domains analysed.

2.2.1 RFID Applications

Radio frequency identification (RFID) is a mature technology used for identifying, monitoring, locating and tracking any physical objects. The main components of a RFID system are the RF tags which are attached to physical objects, the RFID readers (which incorporate also some antenna) and the base station where all the data is collected. The RFID technology boosted the CEP, as companies that deployed RFID systems needed methods for translating raw RFID data into meaningful information that could be further processed by enterprise applications such as access control, document tracking, smart box, logistics and supply chains, healthcare, etc. [23]. It provided a good field for improving CEP methods as it requires less complexity then financial domains from the analytical point of view and less technological overhead for real live deployments compared to sensor network domain. However, RFID systems present other challenges for data processing and information extraction: high volume of streaming data, as each RFID reading is considered an event; raw RFID events encode their semantics and special transformation and aggregations methods are needed [9]. The event streams from large deployments of RF tags can get up to thousands of events per second or even higher, challenging the CEP systems which have to filter and associate events to match predefined patterns [8].

In [8], the authors describe an event model, used by the SASE event processing system, with the purpose of defining a language and a query-plan execution for increasing the performance of scenarios where there is a high amount of RFID events and where large processing windows are also required. The input for SASE was an event stream, which represented an infinite sequence of atomic and instantaneous events. The event language used had the capabilities to filter individual events, associate multiple events based on time and value constraints and specify the output of a query as combination of the attributes of the associated events. The usage of the language is illustrated for applications that handle misplaced inventory for retail management and monitor medication for healthcare. The system has several limitations, including language limitations (no support for aggregates and expressing hierarchies of complex events) and concurrency limitations (events were assumed to be totally-ordered).

Another approach is presented in [9], where the authors describe several steps that should be followed in CEP for RFID applications:

- **Data filtering** is the first step in processing raw (or primitive) RFID events and imply a *low level filtering* and a *semantic filtering*. The low level filtering refers to cleaning methods for removing duplicate readings (RF tag is in the scope of a RFID reader for a long time) and other erroneous readings or false positive readings (wrong RF tags, etc.); it is generally performed at the middleware level, when data is collected. Semantic filtering methods are used for extracting data on demand and interpreting the meaning of RFID raw events. An example of semantic filtering would be associating the meaning of “put on the shelf” and “taken off the shelf” for objects in a retail building, for the first and last readings of an RF tag by the same reader.
- **Data transformation and aggregation** are following after data filtering. Transformation methods refer mainly to updates in the database containing the state of an object (e.g. updating the location of an object, which can be either geographical or symbolical – a shelf, a room). Aggregation is the formation of relationships among object, such as a containment relationship that identifies when more objects are contained by another object (smart box).
- **Complex RFID event definition** is done through event constructors which are specifying the constituent events of the complex events, which can be either raw (primitive) RFID events or other complex events. Two types of event constructors are identified: non-temporal and temporal, where the latter contains order or temporal constraints.

- **Processing of non-spontaneous events** refers to detecting the non-occurrence of an event. The methods proposed is based on pseudo-events that are generated to actively trigger the querying of non-spontaneous events. These pseudo-events are basically the temporal constraints transformed into first class objects in the event detection phase.

The work presented in [9] is extended in [10] through an application of CEP for managing critical situations in hospitals, based on RFID technology. The language used for event descriptions is extended with special operators needed for the scenario presented and the implementation is based on a different rule engine.

2.2.2 Sensor Networks

Sensor networks are composed of sensor nodes with wired or wireless connectivity. The sensor nodes can have several attached sensors, monitoring different properties such as temperature, humidity, light, pressure, wind speed, etc. Other components such as microcontrollers or memory chips allow different processing capabilities, but generally sensor nodes are referred to as constrained devices from the point of view of computational speed, memory, communication bandwidth, or energy consumption. The increasing interest in sensor networks has led to important technological advances that allow wireless sensor networks to manage from few hundred nodes to several thousand nodes. In these conditions, CEP plays a very important role in discovering meaningful information from the vast amount of produced data.

In sensor networks, besides the notions of simple and complex events we also have the notion of *observations* which are the raw outputs of sensors, while an event is something that can be monitored or detected and is of interest for the application. There are two main approaches for CEP in sensor networks: centralized and distributed processing. In the centralized approach, observations sampled by the sensors are transmitted to a gateway where powerful machines can be used for processing. However, this approach presents several drawbacks, including increased energy consumption due to continuous data transmissions and delays in event detection caused by transmission latency. The second option, distributed processing, allows observations to be evaluated on the nodes (up to some degree of complexity), and then transmitted to gateways as events, reducing thus communication overhead. Distributed processing is important for reducing communication overheads, increase availability and overall performance; however problems may arise regarding event detection accuracy, as observations are processed separately.

Some of the methods adopted in [24], and which we consider as good practices for **distributed processing** in sensor networks are:

- **Data centric storage** – data is mapped to different locations according to its semantics and the activity in which it is used. For data lookup, multi-level hashing functions to map data to physical storage nodes are used. Data robustness is achieved by replicating data in multiple physical nodes in order to avoid node failure due to disasters, damages, energy shortage, etc.; some consistency issues may appear and most common is to assure weak consistency in order to avoid heavy work load.
- **Data caching** – providing multiple copies of the most requested data around the frequently queried nodes. The trade-off between response time and consistency overhead must be taken into account.
- **Group management** – localized cooperation among sensor nodes for (1) providing higher reliability for data collected by sensors from the same area, (2) detecting anomalies in sensor measurements or damaged sensors. Groups can be formed dynamically, depending on the task which must be accomplished, or statically, at system deployment, when the application tasks are known in advance.
- **Publish/subscribe** mechanism for event subscription.

A volcano monitoring application that uses sensor networks for event detection is presented in [5]. The challenges faced in using sensor networks for studying volcano activity included high data rates, high data accuracy and sparse arrays with high spatial separation between nodes. Moreover, sensor observations are relevant only for discrete events, such as eruptions, earthquakes and tremor activities and the seismic and acoustic sensors used for monitoring, acquired data faster that it could be transmitted. Therefore, data collection for the sensor deployment described in [5] was triggered by the detected event, adapting thus to the communication bandwidth constraints. Event detection was performed in a distributed fashion on each node, base only on local observations, by applying an event detection algorithm based on averaging observations for short and long periods of time and then sending a message to the base station when specific thresholds were exceeded. If more nodes reported events at the same time, the gateway initiated the data collection procedure where each node sent the sampled observations for the specified period of time.

High accuracy is often needed for the sensor observations as the models used in data processing rely on the correctness of the input data. This was also the case for the volcano monitoring application, which needed large amounts of very precise data, and therefore imposed high demands on communication and data transmission. A solution for handling imprecise data and modelling uncertainty is to use fuzzy logic for event detection. In [7] fuzzy logic is shown to improve event detection accuracy compared to well know classification algorithms (Naïve Bayes, decision trees). However, the method has not yet been applied on a real sensor testbed, and the complexity introduced by a larger rule base needed in fuzzy logic might not be suited for the capabilities of sensor networks.

2.2.3 Financial Applications

The financial domain expresses high interest in CEP techniques as the amount of streaming data relevant for diverse financial services is constantly increasing and traditional methods cannot keep up with these changes. An interesting observation done by a research group is that the average lifespan of a trading algorithm can be as short as three months due to the increasing efficiency in current markets supported by the CEP technologies [25]. Applications for the financial domain include: fraud detection in credit card transactions, algorithmic stock trading, real-time profit and loss analysis. Source of data stream can include stock tickers, news feeds, sensor readings, operations, etc.

Specific characteristics of financial services that should be fulfilled by any application are identified in [11] as the followings:

- high availability of systems (i.e. 24 hours 7 days a week) in order to avoid any data loss; this requires special online update methods.
- efficient handling of both short and long time windows, from seconds to days/weeks/month.
- high number of heterogeneous data sources
- event classification and dependencies
- possibility to express hierarchies of complex events, which means that an event detected by the system must return as an input to the system

In [12] the authors define a three step cycle followed in business applications that is composed of a monitoring step, a management step and mining step (M^3 cycle: monitor-manage-mine). The monitoring is done in real time for predefined indicators and answers continuous queries for detecting the patterns of interest. The results of the monitoring step are used in the same time for managing and performing business actions such as sending alarm messages in case of fraud detection, reporting the profit and loss parameters, etc. The mining step is applied on stored offline data in order to improve the parameters used in monitoring and to detect new interesting event patterns.

The importance of the high performance of systems in terms of low latency and high throughput required in the financial domain and especially for the algorithm trading applications is considered in [26]. The authors developed a prototype based on reconfigurable hardware devices (FPGAs) that showed to outperform the PC-based implementation. The advantages of this approach are given by the reduced overhead of having operating systems, the ability to handle data streams in parallel with lower computational costs, and the higher resolution system clock available on such hardware components compared to PCs. Such implementation may be appropriate for the monitoring step from the M^3 cycle, in order to increase the system performance; it would, however, increase the complexity of integration.

2.2.4 Discussion

In this sub-section we discuss several characteristics of CEP applications for the three domain presented throughout the section.

Ability to give instantaneous reposes refers to the system performance in term of response time. It is a very important requirement for RFID applications, especially for alarm detection scenarios (e.g. in a smart hospital erroneous procedures can endanger life; in a shop, product theft must be immediately identified to be of any value). For financial applications, real time responses are also one of the first concerns (e.g. algorithmic trading), especially as the event arrival rate often becomes higher than the processing capabilities of the system. For the sensor networks domain the importance of instantaneous responses depends on the scenarios where they are applied (e.g. fire alarms would need instantaneous responses, while for volcanic monitoring the constraints can be reduced to minutes rather than seconds).

Distributed processing is needed for sensor networks applications as it reduces energy consumption and communication overhead and it is also applicable as the nodes in the network can provide computation capabilities. For the financial application distributed processing might be harder to implement due to privacy issues and high demand for consistency, while for the RFID domain it is less required.

Logging and analysis of data streams events detected is especially important for financial applications as it is the basis for finding new improvements in the business logic. Some of the applications of sensor networks might also require logging of data for later analysis (e.g. environmental studies), while RFID applications do not present an evident need for this.

Heterogeneity of data sources is generally very high for financial applications where new data sources are often added for finding new methods for improvements; it can be considered a medium requirement for sensor networks, as most of the data comes from sensors and the heterogeneity is given by the sensor types, while RFID applications present a low heterogeneity of data sources.

Learning methods can be successfully applied for the financial and sensor networks domain, while the RFID data presents a lower interest. An interesting aspect is that the systems analysed in each of these domains have not presented any methods for direct integration of learning algorithms with the incoming data streams, assuming separate processes.

Finally, both *volume* and *velocity* are generally high for all three types of applications.

A summary of the aspects discussed is given in Table 2. The importance of satisfying the characteristics described is given using three qualitative levels: low, medium and high.

Table 2 Importance levels of CEP characteristics

	RFID	Sensor Networks	Financial
Ability to give instantaneous responses	high	medium	high
Need for distributed processing	medium	high	low
Need for logging and analysis	low	medium	high
Heterogeneity of data sources	low	high	high
Applicability of learning methods	low	high	high
Volume of incoming data	high	high	high
Velocity of incoming data	high	high	high

3 Critical Overview and Required Improvements

One of the more significant findings to emerge from this study is that few of the research approaches analysed has given a solution for the integration of machine learning or data mining algorithms into the systems presented, for direct support in definition of event patterns. Although massive amount of research has been conducted in areas such as pattern recognition and multisensor data fusion, the systems developed for the applications analysed do not provide a seamless integration with such techniques, but rather consider the human component responsible for defining the complex events patterns that should be monitored and detected. This implies that either domain experts are capable of providing the definition of event patters or that other tools are used externally of the CEP systems in order to discover these patterns. Therefore, an important improvement for applying machine learning algorithms in event-based applications is to develop a framework that would allow easy integration of existing algorithms with event processing techniques. This will require methods for handling both stored and streaming data and should also provide different levels of abstraction (visualisation and testing methods) that would increase the system usability among domain users (which are not necessarily machine learning experts).

Another critical aspect is that the systems built for different applications make particular assumptions (especially the ones using simulated data), such as the quality of data or the reliability of communication channels, that can lead to surprises when such systems are deployed in real world environments. This requires more general CEP platforms that are then applied to real world scenarios and tested for their performance, which would indicate other further improvements that might be needed. Another benefit that could follow from real world experiences with CEP platforms would come from the guidelines and methodologies (lessons learned) that would be defined after several experiments. To the best of our knowledge such general methodologies are currently missing from existing research literature, and they can only be found for specific domains of application as it can be observed also from our analysis.

Related to the issue of general CEP systems is also that of event processing languages, which, as described in this report, vary regarding the type and the number of operators and constructors for complex events. Moreover, as these languages originated from different research areas, such as active DBMS, DSMS and rule engines, they introduce confusions regarding their similarities and differences. It is not clear if any of the language is better than the others or how much each operator contributes to the language expressivity. Therefore a thorough comparison of these languages is needed, in order to identify what are the most important operators that would be sufficient for event description, detection and reaction. Such a study would also answer if a common standardized language for event processing can be built.

In the context of large scale data, CEP is very useful for applications that require real-time or near-real-time response. In contrast to the store-then-process paradigm, the CEP is intended for managing data in motion. The research challenges may arise from three different situations:

- large number of queries
- large number of events
- queries that need large working memory

The first challenge can be addressed by running multiple independent CEP engines with optimizations regarding the distribution of the queries over these independent engines. The second challenge require parallel-distributed platforms capable of partitioning data across processing nodes without the need to process the whole input in a single node for obtaining the same results as a centralized execution. To go into more details, this challenge translate into: (1) distribution of input data so that tuples that must be aggregated or joined are received by the same processing node and (2) flexible resource allocation for demanding computations, given a varying volume of data that must be processed [28]. The last challenge refers to complex queries that need to maintain a large window of events and can be addressed by using a distributed cache.

Last but not least, other research challenges that are worth mentioning are: uncertainty handling and enrichment of events. The first one would increase the performance of CEP systems in front of noisy data and one solution was mentioned for the sensor network domain by applying fuzzy logic methods. Regarding the event enrichment problem, it would help in defining more complex events by adding new information to the basic events by means of computation or by pulling information from external sources (e.g., the web, databases, etc.).

The list of shortcomings mentioned illustrate possible research directions, which can be considered in our future work. In the next section we will present the results of hands-on experiments performed using exiting CEP systems. The experiments have been performed with the scope of familiarization with the systems in

order to better understand the exiting functionalities and define further requirements that can lead to addressing some of the shortcomings mentioned.

4 Event Processing for a smarter city

In this section, we present the results of the exploratory experiments performed using the existing Microsoft StreamInsight CEP system in a specific application context, specifically, that of complex event detection in a smart city. Although CEP is not the main problem discussed in this section, the experiments performed are pre-requisites for dealing with the CEP problem, which will be further addressed in our research.

Smart city scenarios can be complex, combining multimodal data (streams) from several sources. The high level requirements for making a city smarter, as envisioned by IBM in the larger Smarter Planet program[3], refer to the collaboration and coordination between city agencies managing different domains (e.g. water management, transportation, buildings, etc.) in order to be able to optimize the limited resources and to efficiently and effectively deliver city services. Moreover, different technologies may enable smarter cities, such as: communication channels (e-mail, instant messaging, etc.), business rules, data sharing (data models, accessibility) and integration of different sources of data [27]. In another study [29], the classification of cities as smart is made based on 6 criteria: economy, people, governance, mobility, environment and living. Out of these, we focus on smart mobility, which refers to transportation (accessibility, modern transport systems) and availability of Information and Communication Technologies (ICT) infrastructure.

We are investigating scenarios for applying ICT infrastructure to a real use-case city. The final goal is to find correlations in different data sources related to transportation and for this, the first step is to identify the data sources and run preliminary experiments for analysing the data. Future steps are discussed in the direction of using CEP engines with the patterns discovered and defining a framework for an easier integration of machine learning and data mining algorithms with CEP systems.

4.1 Dataset description

London city was selected as a smart city case study due the fact there are several initiatives, such as the London datastore[4], for providing open data for analysis. The data sources of potentially useful information that we have identified are listed below:

- **Traffic data** (bus schedules and delays, congested roads, bicycle availability status, etc.). Sources: Bing Maps, Traffic for London (Tfl).
- **Weather data**. Sources: Weather Underground, Yahoo! Weather, AccuWeather, etc.
- **Events** happening in the city: Live music, conferences, festivals, galleries, sports, etc. Sources: Eventful.com, upcoming.org, last.fm, zvents.com, socialevents.com.
- **Social media** about the events (micro-blogging and news). Sources: Twitter, IJS newsfeed.

We have started to collect data through the publicly available APIs of the data sources listed in Table 3. After receiving the data through the data sources APIs, custom built adapters are used for storing data in a uniform data structure and/or creating stream adapters for CEP processing, using some of the existing systems (StreamInsight[5] and Esper[6]).

Table 3 Data Sources for London city

Source	Starting day for collecting data	Ending day for collecting data	Update time interval	Feed Format	Memory size of data collected
Tfl Road Disruptions[7]	July 16th 2012	April 6th 2013	5 minutes	XML	27 GB
Bing Maps Traffic API[8]	July 16th 2012	ongoing	5 minutes	Json	2 GB
Barclays Cycle Hire [5]	July 16th 2012	ongoing	3 minutes	XML	34,2 GB

[3] http://www.ibm.com/smarterplanet
[4] http://data.london.gov.uk/datastore
[5] http://www.microsoft.com/en-us/download/details.aspx?id=30149
[6] http://esper.codehaus.org/
[7] http://www.tfl.gov.uk/businessandpartners/syndication/
[8] http://msdn.microsoft.com/en-us/library/hh441725.aspx

Current Weather Conditions[9]	July 16th 2012	ongoing	30 minutes	Json	100 MB
Social Events[10]	July 16th 2012	ongoing	Once per week	XML	3,4 GB
Tweets[11]	December 19th 2012	ongoing	streaming	Json	75 GB

Based on the data collected some of the tasks which can be performed using CEP technologies may refer to: 1) finding patterns for the appearance of traffic disruptions that could be applied by a CEP engine for sending different alarms and (2) discover interesting correlations between cultural events happening in a city, social media and their influence on traffic.

4.2 Analysing the public bike renting system using StreamInsight

As a first step in understanding existing CEP tools and gaining insight into some of the collected data, we analysed London's bike renting system data. The Barclays Cycle Hire XML feed contains the name, location, co-ordinates and maximum number of docking points for all operational Barclays Cycle Hire docking stations. It also contains the number of available bikes (excluding locked or faulty bikes), and number of available docking points. The feed comes directly from the Service Provider's information database and is updated every three minutes.

The total number of bike stations analysed is 541. For each of them we get the current status of available bikes and empty docks every 3 minutes. This results in a stream of $25.9*10^4$ rows per day, or $9.3*10^7$ rows per year. In order to analyse the usage of bikes during one year, to discover possible usage patterns or season discrepancies, one can easily see that this can be considered as a good applicability of event processing systems.

4.2.1 Data processing using StreamInsight

StreamInsight is a .NET-based platform for the continuous and incremental processing of unending sequences of events from multiple sources. It is a temporal query processing engine that enables an application paradigm where a standing (i.e., potentially infinitely running) query processes these moving events over windows of time. Processing can range from simple aggregations, to correlation of events across multiple streams, to detection of event patterns, to building complex time series and analytical models over streaming data [30].

The scope of our experiment is to connect the bicycle availability feed to the StreamInsight platform. After creating an adapter for the data source, using the Observer design-pattern [31], we run simple queries for calculating the sum of the total number of bikes available at one moment over all docking stations, the average number of bikes available over one hour, etc., with the purpose of getting accustomed with the StreamInsight platform. The two example queries mentioned are expressed below in LINQ[12] (the query language adopted by StreamInsight):

```
var query1 = from win in inputStream.
             TumblingWindow(TimeSpan.FromMinutes(3))
             select new OutputEvent {nbBikes = win.Sum(e => e.nbBikes),
             nrFreeStations = win.Sum(e => e.freeStations) };

var query2 = from win in query1.
             TumblingWindow(TimeSpan.FromMinutes(60))
             select new OutputEvent { nbbikes = (int)win.Avg(e => e.nbBikes),
             freeStations = (int)win.Avg(e => e.freeStations)};
```

[9] http://www.wunderground.com/
[10] http://eventful.com
[11] https://twitter.com/
[12] http://msdn.microsoft.com/en-us/library/vstudio/bb397926.aspx

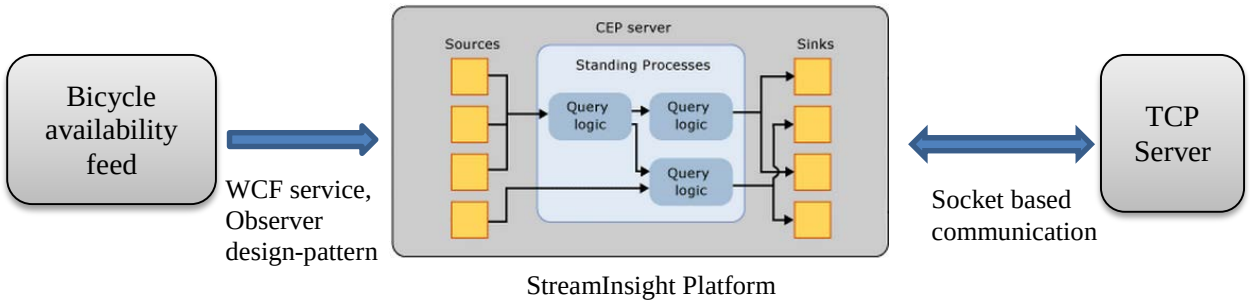

Figure 5. Software architecture of demo application

It can be observed that the time windows used in the example queries are tumbling windows (also known as gapless, non-overlapping hopping windows). For the first query, as the update interval for the bike availability stream is of three minutes, and the windows interval is of three minutes as well, the result of the query will actually compute the sum of available bikes for every update from the stream. The second query is built on the top of the first query, and computes an average over 60 minutes of the sums calculated.

After that we created a small demonstrative application that is updated in real time as bike usage reports are available. The application displays the number of bikes available at the docking stations over the city. The software components of the application are displayed in Figure 5.

The images below (Figure 6 and Figure 7) show the expected pattern of morning versus noon biking paths (i.e. in the morning people move from periphery to the centre, leading to empty docks at the periphery and full ones in the centre of the city; an opposite moving pattern was observed for the afternoon). This can be noticed by the colour codes used for the docking station pointers (described in the images as well).

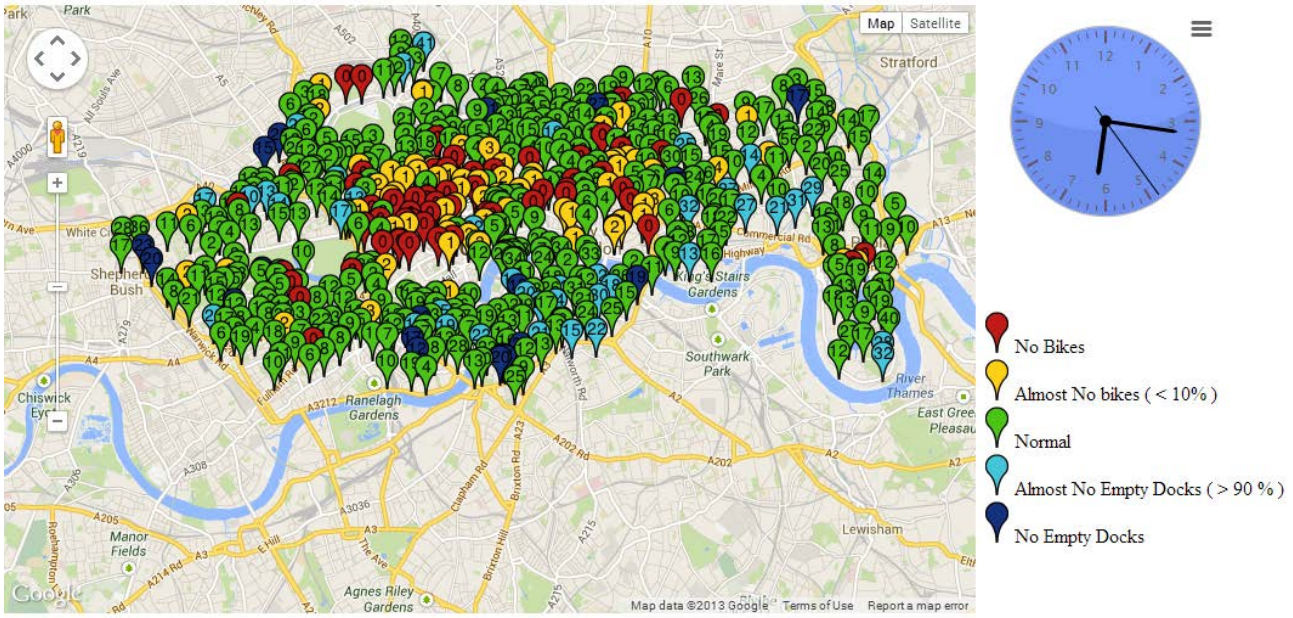

Figure 6. Bike availability at 6 am.

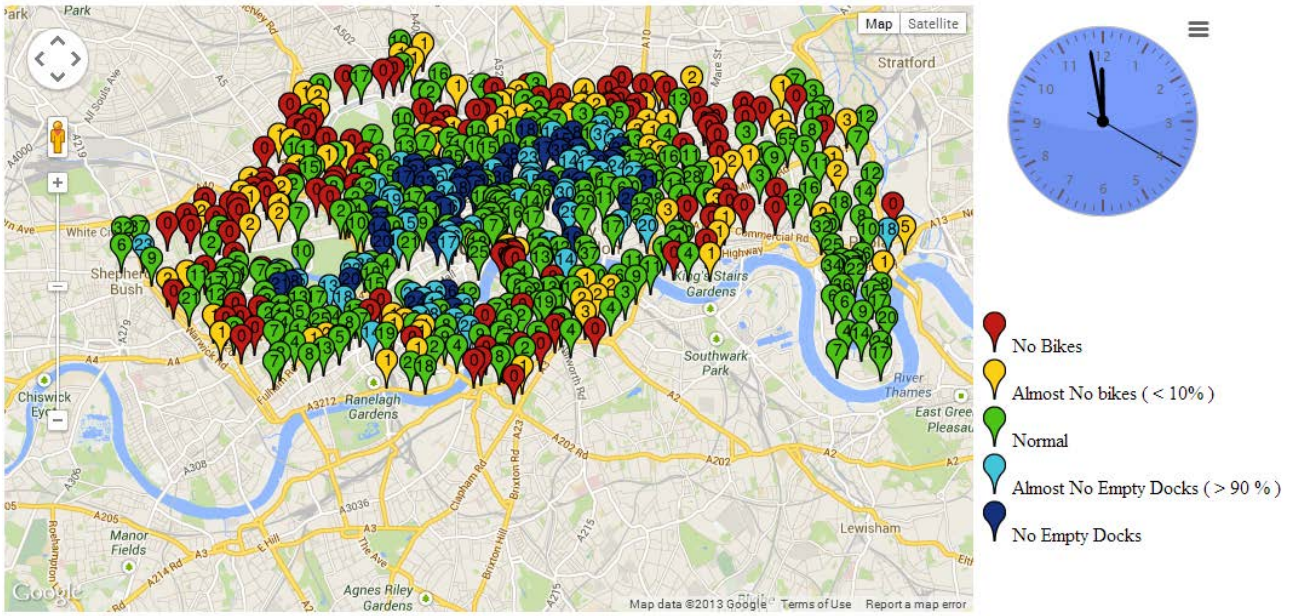

Figure 7. Bike availability at noon

4.2.2 Conclusions

By exploring the data using this simple application, we observed that motion patterns across the city of London can be detected by monitoring the usage of the public bike system. StreamInsight is a tool that is useful for building the backend of such infrastructure and handles well the amount of data we are dealing with. On-going work consists of complementing StreamInsight with data mining operators.

4.3 Discovering social event popularity based on Tweets using Esper System

The scope of this experiment is to determine the popularity of social events happening in the city by correlating tweets to each of them and determine how popular an event was. We consider the popularity directly depended on the number of tweets we can correlate. The cultural events are described by time, location, performers and can be of more categories (e.g. music concert, arts, sports, etc.). Each event has a number of 59 fields from which, for the first experiments, we are interested only in: event title, description, start and stop time, geographical coordinates, id and type of event.

Before starting to correlate tweets to events we performed data pre-processing, which includes parsing the data from the input stream and handling missing values. Since events were registered on the website by the date they were added, we sorted them chronologically by starting time of event. Some events have missing ending time (approximately 27% were missing their ending time); for these we calculated the median of the event duration for each category of event and approximate the end time using this value.

The next step was to process the event and tweet text fields. First of all, we removed redundant characters that are not letters such as: coma, numbers, parenthesis and html syntaxes, all the words with length lower than 3 characters and redundant words such “that”, “this”, “and” and changed all letters to lower case for faster word matching. The resulting dataset consisted of a number of 4.697.200 tweets and 22.704 events collected in the period from March 6th to April 11th 2013.

4.3.1 Data processing using Esper

NEsper is a .NET version of Esper [32] with some customizations that are specific to the .NET CLR environment, so throughout the rest of the document we will refer to it as Esper. Esper is an Event Stream Processing (ESP) and Complex Event Processing (CEP) system. It is designed for high-volume of data where it is hard or impossible to store everything in a database, therefore, it satisfies the need of real time processing. Esper is used in several areas such as finance, fraud detection, medicine, where decisions need to be taken as fast as possible.

The principle of the Esper system is that it allows registering queries in the engine and creates a listener class that will be called if incoming event matches inserted query. The query can contain timeline windows, filtering, aggregation and sorting and they are expressed using a dedicated event processing language (EPL)

Another functionality of the Esper system is to generate a new stream as combinations between two or more input streams. The EPL statement used in our applications is the following:

```
select * from pattern [every Event -> every Tweet
            (Event.Stop_Time-Tweet.Time>0)]
```

which is similar to an inner-join statement from classical database management systems, where the join condition is represented by the time constraint. In order to obtain only the combinations that overlap over time, we use a pattern-based event stream structure, on which we specify the time constraints we want to impose. Therefore, the event stream generated will contain only the event-tweet pairs for which the timestamp for the tweet is between the start and stop time of the event or two hours before the event (the start time of the event is altered with two hours before the initial start time).

4.3.2 Results of the experiment

After we obtain all combinations between tweets and events we created a method that is called trough the Esper engine and determines if the tweet is correlated or not with the event. To determine the degree of correlation between a tweet and an event we define a correlation coefficient (*C*) as:

$$C = 0.5 * P + 0.25 * W + 0.125 * L + 0.125 * B,$$

where P = 1 if tweet text contains the performer's name of the event, W is calculated by the number of words matching the tweet text and event title divided by the total number of words in the event title, L=1 if the location name is found in the tweet text and B = 1 if the text contains the bio description of the performer. The weights of these parameters are set based on a common sense understanding of their meaning.

A low value of *C* is understood as a low degree of correlation between the tweet and the event analysed. In order to analyse the performance of *C* we manually evaluated a random set of associations of tweets and events. We first set a threshold value for *C* to 0.25 and then we randomly selected 100 associations of tweets and events with correlation coefficient higher than 0.25 and manually verified if correlations have sense or not, by reading the tweet and the event title and giving 1 point for the correct correlations and 0 for the incorrect ones. This procedure was executed by 2 annotators, after which we calculated inter-annotator [33] agreement to check how good was the evaluation. The inter-annotator agreement, or Cohen's kappa coefficient is described by next equation:

$$k = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)},$$

where Pr(*a*) is the relative observed agreement among annotators, and Pr(*e*) is the hypothetical probability of chance agreement. We obtained Cohen's kappa coefficient of 0.661, which translates to a "substantial" level of agreement [33].

The next step was to analyse the annotated associations, considering only those for which the annotators gave the same score. First observation is that the average of *C* values for correct associations is higher than the average of incorrect associations: 0.52 and 0.36. Further analysis of the performance of C is done in Figure 8 and Figure 9.

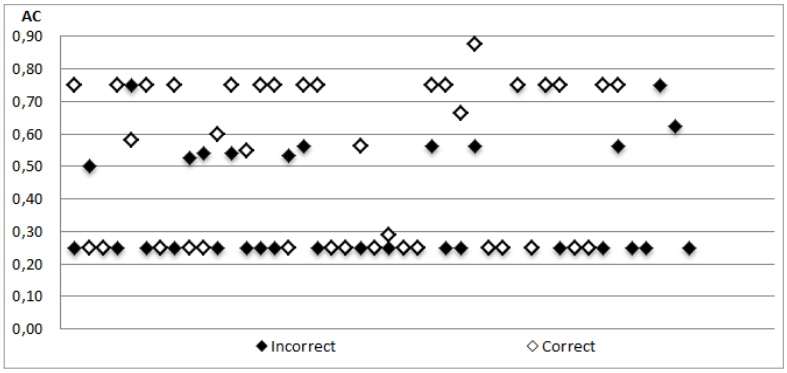

Figure 8. Values of C for the associations of tweets and events manually evaluated (the empty dots have been annotated as correct, while the full dots as incorrect)

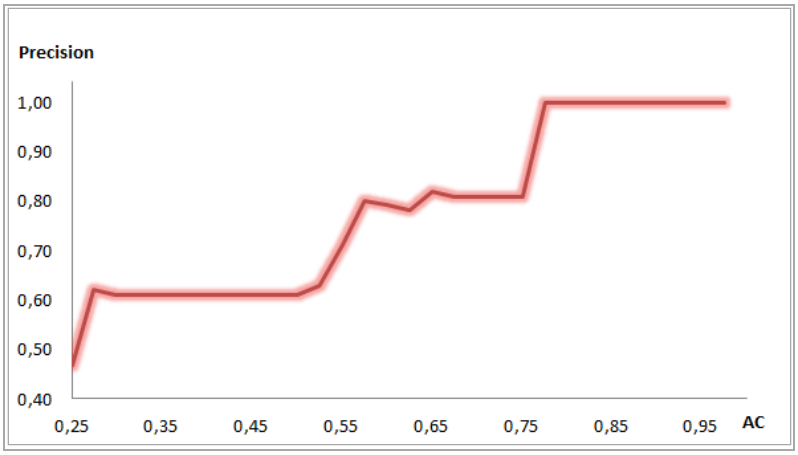

Figure 9. Precision of associations for different values of *C* (we can observe that for C > 0.75 the precision improves dramatically)

When data processing is finished, we register it to the output <name>.txt file in the next form: first we insert the event fields and after each event a list with the correlated tweets. The data representation (Figure 10) is customizable by the user by checking the fields he wants to register to file (Id, text, coordinates, type etc.).

```
**********************************************************************************
Event Id: E0-001-054837595-5
Event Title: dj fitz
Event Description: power dj fitz spin mixing bouncy melodic rough smooth impossible aback eclectic
mix styles beats gypsy tones rhythmically driven bass lines kind music dance heard ridley road market
bar laid back atmosphere east vibes smashing djs delicious street food pretty awesome staff ahem top
early avoid queues don cold
Event Coordinates: 51.5481258 -0.0723959
Event Start Time: 3/22/2013 6:00:00 PM
Event Stop Time: 3/22/2013 6:00:42 PM
Event Types: music singles_social
----------------------------------------------------------------------------------
Tweet Id: 309269851921334273
Tweet Text: dj fitz today
Tweet Coordinates: 0.18317658 51.47360666
Tweet Hastags:
Created At: 3/6/2013 11:50:53 AM
**********************************************************************************
```

Figure 10. Representation of results

In this experiment we made a basic event-tweet correlation to find the popularity of events. Although there are things left for improvement, this experiment has allowed us to gain experience with the Esper system and prepared the ground for further experiments. First of all we would like to increase and select a larger number of relevant stop-words for filtering. By selecting more stop-words we aim to improve the accuracy of tweet-event correlation. Another step is to define more complex rules by involving other data from event and tweets. For example we can search tweets that are only published in nearby location with the event, decreasing the processing time.

5 Conclusions

This deliverable has first given an overview of the existing research for the CEP domain by analysing the two types of languages used in describing and querying events and discussing the characteristics of three main application areas where CEP methods have been applied: RFID applications, sensor networks and financial applications. Then, a set of collected data for a smart city scenario targeting London has been described. The data has been used to develop a bike-renting system monitoring application using the StreamInsight CEP tool and to detect correlations between tweets and public events using the Esper tool.

References

[1] Stadtmüller, S. et. al. Best practices on how to deploy tools on large-scale infrastructure. Planet Data Deliverbale 5.2, 2012. http://wiki.planet-data.eu/uploads/8/8d/D5.2.pdf

[2] Stonebraker, M., et al., The 8 requirements of real-time stream processing. SIGMOD Rec., 2005. 34(4): p. 42-47.

[3] Cugola, G. and A. Margara, Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys* (CSUR), 2012, 44.3: 15.

[4] Etzion, O. and N. Peter, Event Processing in Action. 2011: Manning Publications Co.

[5] Werner-Allen, G., et al., Deploying a wireless sensor network on an active volcano. Internet Computing, IEEE, 2006. 10(2): p. 18-25.

[6] Wittenburg, G., et al., A system for distributed event detection in wireless sensor networks, in Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks2010, ACM: Stockholm, Sweden. p. 94-104.

[7] Kapitanova, K., S.H. Son, and K.-D. Kang, Using fuzzy logic for robust event detection in wireless sensor networks. Ad Hoc Networks, 2012. 10(4): p. 709-722.

[8] Wu, E., Y. Diao, and S. Rizvi, High-Performance Complex Event Processing over Streams, in Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data2006, ACM: Chicago, IL, USA. p. 407-418.

[9] Wang, F., S. Liu, and P. Liu, Complex RFID event processing. The VLDB Journal, 2009. 18(4): p. 913-931.

[10] Yao, W., C.-H. Chu, and Z. Li, Leveraging complex event processing for smart hospitals using RFID. J. Netw. Comput. Appl., 2011. 34(3): p. 799-810.

[11] Asaf, A., et al. Complex Event Processing for Financial Services. in Services Computing Workshops, 2006. SCW '06. IEEE. 2006.

[12] Chandramouli, B., et al., Data Stream Management Systems for Computational Finance. Computer, 2010. 43(12): p. 45-52.

[13] Chandy, M., O. Etzion, and R. von Ammon, 10201 Executive Summary and Manifesto–Event Processing, in Event Processing, M. Chandy, O. Etzion, and R. von Ammon, Editors. 2011, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany: Dagstuhl, Germany.

[14] Event Processing Glossary, D. Luckham and R. Schulte, Editors. 2008, Event Processing Technical Society.

[15] Price, M., The rise of the picosecond, in Financial News2011: http://www.efinancialnews.com/story/2011-03-03/rise-of-the-picosecond.

[16] Arasu, A., S. Babu, and J. Widom, The CQL continuous query language: semantic foundations and query execution. The VLDB Journal, 2006. 15(2): p. 121-142.

[17] McCarthy, D. and U. Dayal, The architecture of an active database management system. SIGMOD Rec., 1989. 18(2): p. 215-224.

[18] Chakravarthy, S., et al., Composite Events for Active Databases: Semantics, Contexts and Detection, in Proceedings of the 20th International Conference on Very Large Data Bases1994, Morgan Kaufmann Publishers Inc. p. 606-617.

[19] Chakravarthy, S. and D. Mishra, Snoop: an expressive event specification language for active databases. Data Knowl. Eng., 1994. 14(1): p. 1-26.

[20] Adaikkalavan, R., Chakravarthy, S. SnoopIB: Interval-based Event Specification and Detection for Active Databases. Data and Knowledge Engineering, 59(1), 2006.

[21] Anicic, D., et al., EP-SPARQL: a unified language for event processing and stream reasoning, in Proceedings of the 20th International Conference on World Wide Web2011, ACM: Hyderabad, India. p. 635-644.

[22] Barga, R.S., et al., Consistent Streaming Through Time: A Vision for Event Stream Processing, in 3rd Biennial Conference on Innovative Data Systems Research (CIDR), G. Weikum, J. Hellerstein, and M. Stonebraker, Editors. 2007: California, USA. p. 363-374.
[23] Luckham, D. and M. Palmer, Separating Wheat from Chaff, in RFID Journal2004: http://www.rfidjournal.com/article/view/1196.
[24] Li, S., et al., Event Detection Services Using Data Service Middleware in Distributed Sensor Networks. Telecommunication Systems, 2004. 26(2-4): p. 351-368.
[25] Bates, J., Secrets Revealed: Trading Tools Uncover Hidden Opportunities, in FixGlobalTrading2011, FixGlobal: http://fixglobal.com/content/secrets-revealed-trading-tools-uncover-hidden-opportunities.
[26] Sadoghi, M., et al., Efficient event processing through reconfigurable hardware for algorithmic trading. Proc. VLDB Endow., 2010. 3(1-2): p. 1525-1528.
[27] Wang, Q., Meegan, J., Freund, T., Li, F.T., Cosgrove, M.: Smarter City: The Event Driven Realization of City-Wide Collaboration. 2010 International Conference on Management of e-Commerce and e-Government. 195-199 (2010).
[28] Gulisano, Vincenzo Massimiliano, et al. "A big data platform for large scale event processing." Ercim News 89 (2011): 32-33.
[29] Giffinger, R., Fertner, C., Kramar, H., Kalasek, R., Pichler-Milanovic, N., Meijers, E.: Smart cities Ranking of European medium-sized cities. , Vienna, Austria (2007).
[30] Krishnan, R. R., Goldstein, J., Raizman, A. A Hitchhiker’s Guide to Microsoft StreamInsight Queries. http://go.microsoft.com/fwlink/?LinkId=256236
[31] Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. 1994: Addison Wesley Professional.
[32] EsperTech Inc, Esper Reference, Version 4.8.0, 2012.
[33] Cohen's kappa. http://en.wikipedia.org/wiki/Cohen's_kappa (last access date: 28.08.2013)