2014 IEEE International Conference on
Automation Science and Engineering (CASE)
Taipei, Taiwan, August 18-22, 2014

Complex Event Processing for the Internet of Things and its Applications*

Ching Yu Chen, Jui Hsi Fu, Today Sung
Ping-Feng Wang, Emery Jou, Ming-Whei Feng

Abstract—**In this paper, we present a distributed complex event processing (CEP) engine for Internet of Things and its applications. A CEP Engine combines information from a variety of sources. It looks for patterns in these event streams and then responds in real-time. Complex event processing is an effective mechanism to analyze object data, to reason sensing events, and to trigger responding actions for the intelligence of IoT application. A building controlled by a building automation system (BAS) is often referred to as an intelligent building, smart building, or a smart home. BAS core functionality keeps building climate within a specified range, lights rooms based on an occupancy schedule, monitors performance and device failures in all systems, and provides malfunction alarms to building engineering contractors and maintenance staff. The advantages of employing a distributed CEP engine for smart building are demonstrated.**

I. Introduction

With the advances in information and communication, driving new technologies such as security monitoring, health care, and green energy applications, the building development has set off a new wave of smart internet of things (IoT) services by active sensing analysis techniques to integrate more applications meeting user needs.

With the advent of IoT, more and more data will be monitored and collected. And with the maturity of IOT, a lot of useful information are produced. How to store and manage the information to discover the inherent value has become an important issue. Part of these large amounts of data are static descriptions of information, while others are streaming data changing frequently. The off-line data can be processed through distributed computing system, like Hadoop. For streaming data, we need to consider near real time data processing. For example, the brand models of air-conditioning equipment are seldom varied. The indoor and outdoor temperatures are constantly changing, and the amount of change data is very large. For a smart building, through environmental sensing, the air conditioning system can regulate indoor climate, and provide the most comfortable environment with the lowest energy consumption according to the needs of the activities inside. To meet the needs of real-time streaming of information processing, it is necessary to design a data processing architecture enabling immediate response. Considered in the case of very large amounts of data, the system will not only be able to process data faster, but also be able to process large amount of data.

Sections of the paper are organized as follows. The streaming event processing and development will be introduced in the second section II. The third section will describe the proposed distributed complex event processing architecture. This architecture will be applied to two application scenarios in the fourth section. The fifth section is the conclusion and future directions.

II. Related Work

Traditionally the data processing depends on database management systems (DBMSs), due to the need for more immediate responses in financial stocks, environmental sensing and traffic monitoring applications, it calls for requirements of data stream processing (Information Flow Processing, IFP) [1]. DBMSs would have data stored in advance, ordered or marked in time, then waited for user needs to retrieve data for computing or processing; receiving data and processing data are not at the same times. But IFP application will process immediately as soon as the data are received, such as the consumption anomaly detection scenario [4]. In other words, DBMSs are suitable for non-immediate real-time response, and the need for immediate reaction should be considered IFP. If the reaction time affect the outcome of the application is enormous, one needs to design appropriate IFP system to handle data.

In response to the needs of data stream processing, a number of different systems has been developed now. The architecture [5,6,7], grammar rules of languages [8,9,10], data models [11, 12] as well as processing mechanisms [13, 14] are different in these systems. It can be divided into two categories : data stream processing model) [2] and complex event processing model [3]. Data streaming processing model integrates streaming data from different sources into a new output streaming data . These output streaming data can be stored in the database with DBMSs, and information to support common retrieval method can be found. The complex event processing model processes data stream by observing events to filter or calculate the corresponding output. Here an event may contain sensing data from different sources or may be a new event merging reprocessed data with new data. The data processing method may become very complicated. In recent years, the development of a grammar defined events to help the user simply the event definition. The complex event processing model having better processing power in handling time relationship among different events plays well in IoT environment. In IoT applications, data are collected from

*This study is conducted under the "Advanced Sensing Platform and Green Energy Application Technology Project" of the Institute for Information Industry which is subsidized by the Ministry of Economy Affairs of the Republic of China.
{chingyuchen, rayfu, today, pfwang, emeryjou, mfeng}@iii.org.tw

different sensors into the system, not necessarily the same frequency or in synchronization, so there are advantages of the time associated method with complex event processing model approach to solve these problems.

In IoT applications, such as accident detection considering security and response time, it is not appropriate to collect all the data into a central server to process. Grell [15] suggests using different methods for streaming data under various scenarios. Wang [16] proposes Proactive Complex Event Processing (Pro-CEP) to process large scale traffic data. But as soon as there is system failure or network disconnection in the centralized processing, it could cause catastrophic crisis. Therefore, if the large amount of data can be preprocessed and only the intrinsic information being sent to back-end to be analyzed, then not only the speed will be increased but also the processing burden be reduced.

This paper proposes a distributed (client-server based) complex event processing architecture. This complex event processing engine can be embedded into a light power computing capacity gateway to pre-processing and filtering simple events. It will not only be able to reduce the burden of network traffic, the amount of back-end computing and storage capacity, but also be able to handle security problems of network disconnection.

III. Distributed Complex Event Processing Architecture

In IoT environment, raw data are often generated by devices in very high frequency. In order to extract meaningful information, we have to perform some preprocessing work on the raw data. Although traditional centralized CEPs also have ability to process and extract information from data, the centralized architecture is unsuitable for IoT environment. A centralized architecture means devices have to send large amount of raw data to central server. However, in IoT environment, network quality could not be always good enough to provide sufficient bandwidth. Besides, large amount of sensors make the central server bearing high computation loading.

With the help of our client-server based CEP architecture, we can easily separate the detecting procedure into preprocessing part and reasoning part. By doing this, not only the network consumption but also computation loading will be reduced. For example, in the case of detecting the abnormal condition of "the air-condition still turned on during the off-work time", we do not really care about the exact power consumption by the air-conditioner. Instead, we just want to know whether the meter values of air-conditioners are in the turned on status or not during the off-work time. In this case, we don't have to send raw meter data to CEP server. We can separate the process into two parts to apply our client-server based CEP. For preprocessing part, because we only care about the off-work time meter values, we firstly filter out the work time data in our gateway-side CEP. After filtering out irrelevant data, we should also determine if the air-condition's meter values are in the turned on status. In other words, the gateway-side CEP will only send warning message to server-side CEP when the meter values show that air-conditions were turned on in off-work time. For reasoning part, the server-side CEP collect the warning message and compare to other gathering data to make sure that the warning message is not just a false alarm which is affected by other event. Afterwards, the server-side CEP could send a warning message to building manager or send a shutdown command to air-condition to avoid energy wasting.

By using the client-server based CEP, the core-engine loading will be reduced tremendously, since we separate the work loaded into gateway-side to extract the useful information from raw data. Moreover, server-side CEP could focus on dealing with semantic reasoning jobs such as correlating between warning messages.

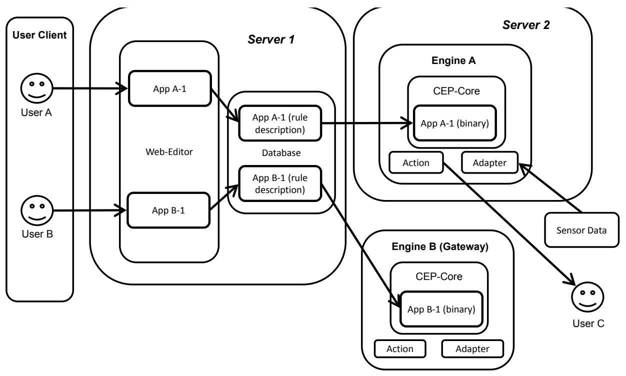

Figure 1. Client Server Architecture

Figure 1 shows the client server architecture of our CEP. Similar to many programing language, there are two stages in our development procedure, which are design time and runtime. In design time, we provide a web interface to do some works such as designing rules, store designed rules, managing CEP engines and managing accounts. As Figure 1 shows, both user A and user B use the web-editor to design application A-1 and B-1 respectively. They can freely change and update their own rules in their applications without worrying about any collision between applications, since each user has their own workspace and we store each application in separated instance. After users finish their design work, they can deploy their application into the CEP engine they want. In this example, user A decide to deploy his application into engine A in server 2 and user B decide to deploy his application into engine B in gateway.

After design time works are completed, users could move to runtime stage. In runtime, users can start engine via web interface to begin the data gathering work. Take the server 2 as example, after user A start engine A, sensor data will flow into engine A through adapter gate. The engine will deal the data with series of rules which are defined by user A, then send notification message to user C through action gate. This is a simplest case. In more complex case, adapter may receive event from action gate in other engine to accomplish the communication between server-side CEP and gateway-side CEP.

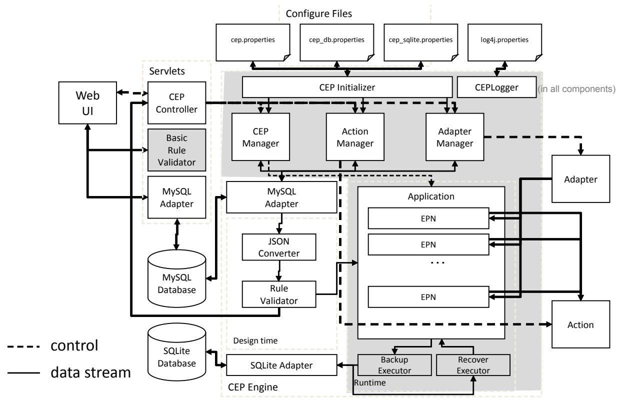

Figure 2. System Architecture

Let us look further inside of each component. The component with gray background in Figure 2 represents the CEP core. All components on left hand side in this figure constitute the web interface.

A. Web Interface

Users could manipulate CEP via the web interface. The first type of work is to remotely manage CEP engines. Since we may have many CEP engines in one test bed, we should have a convenient interface to control all CEP engines. In our CEP, we use RMI and JMX technologies to accomplish the remotely control mission. With the web interface, we can provide commands to start, pause or stop any engine. Afterward, CEP controller will interpret our commands to the corresponding CEP engine. Users can use the same way to turn on/off the adapter gates or action gates to control the flow path out of CEP engines. If users need extend the numbers of engines, they can dynamically add them by register the IP addresses into CEP controller.

The second type of work is to edit rules. We provide an easy-to-lean flow-based graphic user interface. Users could add or modify rules without writing hard code. We let users use drag and drop technique to add and reorder steps. Before users want to deploy the application, basic rule validator will examine the syntax of the rules. If the rules don't pass the validator, users will get error message and they should correct the rules before they retry deployment. This validating feature could insure the designed rules able to be converted to workable instance in CEP engine.

The third type of work is to save rules into database. Because the web interface may provide services to many users at the same time, database is the best way to manage rules generated by different users. Before users deploy applications on to engines, they can store unfinished rules into database. We let all users has their own working spaces. In their working spaces, they can only see applications which they have access permission to read and write.

B. Adapter and Action

The adapter and action components are on the right hand side of Figure 2. Adapter and action are two kinds of gates for communication of CEP engine. Adapter is responsible for adapt stream data into CEP engine and action is for sending results to end points. The adapter directs stream data to Event Processing Networks (EPNs) which contain the rule set inside the application. After the processing of EPN the result will be send to action gate. There are many other kinds of adapter and action gates such as database, socket, REST, mail, JMX and SMS, through which we can adapt stream data from many different kinds of sources and send results to many different kinds of targets too.

C. Operating Mechanism inside CEP Engine

As Figure 2 shows, CEP engines will be started by CEP initializers, which read series of configuration files to setup hardware. CEP initializers will activate three managers which are CEP Manager, Action Manager and Adapter Manager. These managers are to interpret commands from CEP controller and do the corresponding jobs such as initiating new instances or change instances' status. These manages are also to read status logs from each instance and send back to users.

An application may contain many rule sets. We call a rule set as Event Processing Network (or EPN in short). In EPNs, users can use different kinds of modules to compose a processing progress. There are mainly four kinds of modules which are Producer, Consumer, Channel and Event Processing Agent (or EPA in short). The Producer's job is to connect to adapter gate and let stream data flow into EPN. The Consumer's job is to connect to action gate and let stream data flow out from EPN. The Channel's job is to connect different EPNs and let EPNs communicate with each other. Users could concatenate different kinds of EPAs between Producer and Consumer. Different kinds of EPAs have different specialties to process stream data. For examples, Filter module is mainly on filtering out unnecessary data. Aggregation module is mainly on executing basic arithmetic operations with input data. Split module can split out a complex stream data into many groups of data. Compose module in aother hand can compose different stream data into a complex stream data. There is also module with analytics function such as SVM (Support Vector Machine) module. Users can use SVM module to build up a predict function and let stream data be classified into different categories.

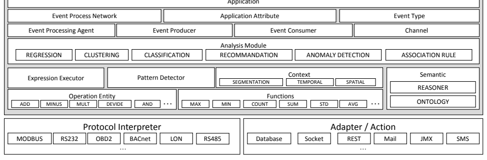

Figure 3. Functional Architecture

D. Functional Architecture

Figure 3 shows the functional architecture of our CEP. Our CEP could be divided into four functional blocks, which are user interface, CEP engine, protocol interpreter and adapter/action. As previous section (Web Interface) shows, our CEP provide web-based user interface to handle jobs such as account/engine management, rule editor and rule storage. These jobs correspond to user interface functional block in Figure 3. In user interface block, the functions could be further organized as rule editor, engine manager and engine controller three parts. The rule editor part handles the rule storage and edit tools. The engine manager manages accounts and monitors all engines' status. Both rule editor block and engine manager block communicate with engines by means of engine controller, which is relied on JMX technology.

The CEP engine block in Figure 3 shows the hierarchies of the processing. On the top of this block is application. We take each processing unit as application in our CEP engine and each application may contain many Event Process Networks (or EPNs in short). An EPN is a group of rule set. An event enters into a rule via event producer. The event will pass through a series of Event Processing Agents (or EPAs in short) and finally enters into event consumer. Each event could bring a series attributes which are defined by event type. The application also haves global attributes called application attributes, which can let user define common attributes shared with all EPNs in the application. Users use channel to connect up different EPNs. There are many types of EPA in our CEP engine. The basic EPA provides fundamental operations of arithmetic and a context to group up relative events. With more complex using, user could adapt extended EPA such as pattern detection EPA or analysis EPA.

In IoT environment, dealing with a lot of heterogeneous protocols provided by different sensors will be a big problem. As protocol interpreter block in Figure 3, our CEP takes the responsibilities to interpret many different protocols and let users could easily connect the devices they need. The adapter/action block in Figure 3 represents the concepts in (Adapter and Action) section. With many sources/targets could be connect to adapter/action gates, users will have high flexibility to change their rules meeting their needs.

IV. Smart IoT Applications with CEP

Internet of Things (IoT) has been attracted in building intelligent offices, houses and cities. In the IoT paradigm, things can be human and environment attached with sensors or electric devices controlled by small computers. They are connected to each other over a network, and particularly their interaction is not involved manually. It is noted that data collected from sensors could be used to analyze environment for further actions. The analytic helps extract relevant information, understand observed behaviors, and recommend suggestions. For example, temperatures in a building could be collected and used to pronounce a fire alarm, and consumed powers measured from power meters in a building could be used to detect abnormal power consumption. In terms of application development, previous work transferred observed data particularly to a centralized server. All computing requirements are dealt with by responsibility of the particular server. Hereafter, two arising concerns while the number of things increases are listed as follows.

- Efficiency: the computational ability is required to analyze behaviors within reasonable response time.
- Scalability: the system developed for IoT should be robust to handle massive incoming data.

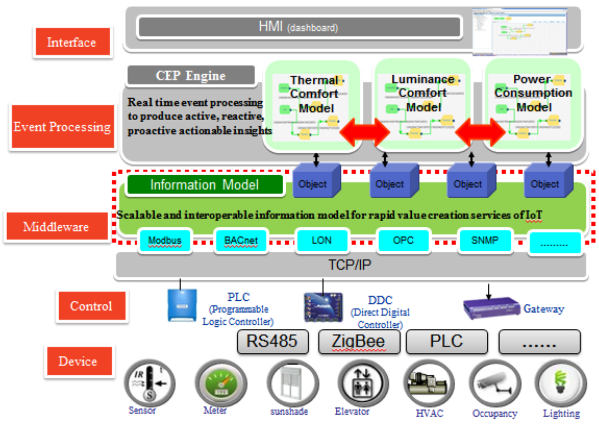

Figure 4. Smart Building with Building Automation System

In this study, the IoT application is a smart building with building automation system (Figure 4). We shall adopt the distributed CEP solution presented in Section III. First, data transferred in the partitioned network are collected by the corresponding CEP engine. Then, the analytic modules in CEP perform further data processes, e.g., filtering irrelevant information, aggregating relative events, classifying target instances, and detecting behavior patterns for predefined actions. The CEP engines will finally convey outputs to designated destinations, e.g. another CEP engine, a centralized server, or a smart mobile device, for further data process and information presentation.

Consequently, there are three advantages of adopting distributed CEP:

- The CEP engines distributed in the network are able to balance work load of processing observed data.
- A new CEP engine can be deployed for increased data volume.
- The network traffic for transferring information between things could be significantly reduced.

In the internet of things, the streaming data rapidly increases complexity. The information of one sensor has been unable to meet the necessary and identify the user behavior. In order to identify users need to detect events and provide immediate response, it integrates quickly to to use a variety of sensing data from the different time zones and geographical. We shall introduce two scenarios of smart buildings which adopt our distributed CEP engines.

In this complex event processing software platform, providing sensing information and real-time event detection monitoring treatment response, including a Network-Based event rule to build flexible complex event processing environment. You can always monitor the real-time streaming event data and event processing engine performance.

In this study we collect through the sensors of temperature, humidity, wind speed, rainfall, luminance, radiation, co2, co, motion, people counter, power meter, etc for a total 1856 property information, together with information related to the environment. We also monitor the energy efficiency of equipment including chiller, cooling tower, ice tank, pump, air side, lighting, sunshield, elevator, etc. for a total of 1,030 properties information.

In the smart building applications, we employ complex event processing engines to monitor the status of sensors and management of devices. By the client-server complex event processing platform, the device side (CEP client) event rules can be defined to control the behavior of the device events for fast response, and capture critical characteristics data (eigenvalues) to be sent to be analyzed in the back-end CEP server for more in-depth information. In the smart building applications, we monitor the operational efficiency of the building 's air-conditioning equipments by CEP editing tools, and provide two operational scenarios :

A. *The first equipment operational performance monitoring rules:*

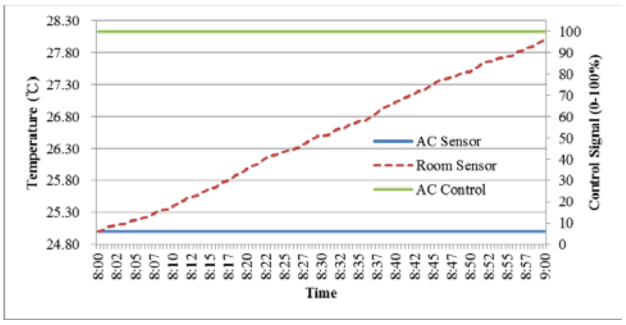

Figure 5. The first abnormal event

Based on air temperature and return air temperature in the air-conditioned area, analyze the sensor has displacement phenomena. For example, when an air-conditioning equipment continued operating in high-speed for 1 hour, but its operational area return air temperature and air temperature are still different more than 3 ℃, the displacement sensor rules will detect the abnormal event and send e-mail notification to equipment maintenance personnel to check the sensor values in order to save energy .

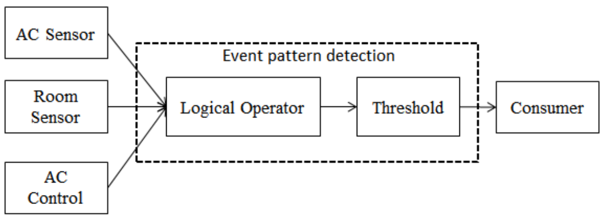

Figure 6. The processing of first equipment operational performance monitoring rules

We use complex event engine editing tools to edit this rule by logical operation and threshold component elements to associate logic to reach set this rule. Three producers are used to receive external signal sensing data; AC Sensor producer receives air conditioning return air temperature, AC Control producer receives air conditioning equipment operation and control signal information , Room Sensor producer receives the regional environmental temperature. The external signal sensing data enter Event pattern detection through " Logical Operator " pattern detection agent (PDA) to select the logical operator type and follow by " Threshold " to check the temperature difference between AC Sensor air conditioning return air and Room Sensor of operational zone (< 3 ℃), while AC Control of air conditioning equipment is operating close to full load (> = 95%). Otherwise, send abnormal events sent to consumer and notify management personnel .

B. *The second equipment operational efficiency monitoring rules:*

Based on the operational efficiency of the device to analyze and compare with other for detecting the abnormal failure. For example: when an air conditioning equipment continuously blast at the highest speeds for 2 hours, the monitoring equipment will produce equipment abnormal events, notify maintenance personnel to inspect the device running under full load and immediately fix the problem in order to save energy devices through the mail.

We use complex event engine editing tools to edit this rule by logical operation and threshold component elements to associate logic to reach set this rule. By 2 producer after to receive AC Sensor and AC Control signals, the external signal sensing data enter Event pattern detection runs through " Logical Operator " pattern detection agent (PDA) selecting the logical operator type, follow by " Threshold " checking the temperature drop, and assure normal operation for 2 hours. Otherwise, on behalf of the abnormal phenomenon for two hours, send abnormal signals to the consumer

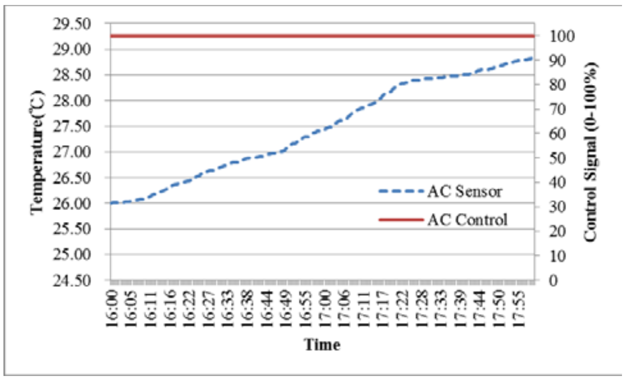

Figure 7. The second abnormal event

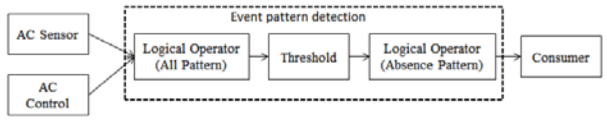
Event pattern detection
AC Sensor
AC Control
Logical Operator (All Pattern)
Threshold
Logical Operator (Absence Pattern)
Consumer

Figure 8. The processing of second equipment operational performance monitoring rules

Smart building scenario can work with the property management industry, the construction of this platform technology into the case, the property management industry through the subsequent maintenance of building energy management system, residents can keep abreast of home electricity usage, or commercial office buildings can automatically regulate the safety of public facilities space such as surveillance, air-conditioning and lighting equipment, situational management, to create low-carbon energy smart buildings, to provide value intelligent management, energy and festival fees; to complex event processing engine based systems.

Sensors in a building's mechanical components will alert engineers through the rule of the complex event engine. Building operators and residential consumers will be able to monitor utility consumption and control heating, lighting, safety, and security systems from offsite. For commercial buildings, that means being able to monitor multiple properties from a centralized property management location, communicate to onsite maintenance crews, and take control of systems in a fire or other emergency.

Complex event processing engine can focus on the wisdom of green energy applications such as: intelligent energy management system integration, active energy-sensing application system integration, the ability to regulate the load platform optimized energy, energy saving, energy scheduling and automatic event detection.

V. Conclusion and Future Works

With IoT getting more and more popular, there is an urgent need of a middleware to connect all devices and let them communicate with each other. Although centralized CEP has the ability to take the role of middleware, the centralized architecture let it become limited and inefficient. To avoid the main drawback of centralized CEP, we develop a distributed CEP which focuses on IoT environment. Distributed CEP would be more suitable than centralized CEP in IoT since users could discard unnecessary data before being sent back to backend server. This characteristic saves a lot of bandwidth in communication and reduces the computation load of backend server.

For simplifying the process to manipulate our CEP, we provide users a web-based visualized editor to define rules. This editor could not only help users dynamically change their rules but also make the designing rule process easier. The web-based user interface could also help users to monitor or deploy a group of CEP engines remotely.

We have built a well-defined distributed CEP architecture to adopt IoT environment. However, there still something we can do to enhance the usability. For example, designing rules is a labor consuming work. In the future, we can adopt some machine learning technique to reach the automatic behavior learning goal. Moreover, designing rule always have to indicate many specific devices or targets. We can add semantic reasoning feature into our rule defining process to allow users use high level concept of rule definitions.

References

[1] Cugola, Gianpaolo, and Alessandro Margara. "Processing flows of information: From data stream to complex event processing." ACM Computing Surveys (CSUR) 44.3 (2012): 15.
[2] Babcock, Brian, et al. "Models and issues in data stream systems." Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, 2002.
[3] Luckham, David. The power of events: An introduction to complex event processing in distributed enterprise systems. Springer Berlin Heidelberg, 2008.
[4] Wen, Jimi Yung-Chuan, et al. "A complex event processing architecture for energy and operation management: industrial experience report." Proceedings of the 5th ACM international conference on Distributed event-based system. ACM, 2011.
[5] Chandrasekaran, Sirish, et al. "TelegraphCQ: continuous dataflow processing." Proceedings of the 2003 ACM SIGMOD international conference on Management of data. ACM, 2003.
[6] Michelson, Brenda M. "Event-driven architecture overview." Patricia Seybold Group 2 (2006).
[7] Abadi, Daniel J., et al. "Aurora: a new model and architecture for data stream management." The VLDB Journal—The International Journal on Very Large Data Bases 12.2 (2003): 120-139.
[8] Stonebraker, Michael, Uğur Çetintemel, and Stan Zdonik. "The 8 requirements of real-time stream processing." ACM SIGMOD Record 34.4 (2005): 42-47.
[9] Anicic, Darko, et al. "EP-SPARQL: a unified language for event processing and stream reasoning." Proceedings of the 20th international conference on World wide web. ACM, 2011.
[10] Bai, Yijian, et al. "RFID data processing with a data stream query language." Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. IEEE, 2007.
[11] Babcock, Brian, et al. "Models and issues in data stream systems." Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, 2002.
[12] Abadi, Daniel J., et al. "Aurora: a new model and architecture for data stream management." The VLDB Journal—The International Journal on Very Large Data Bases 12.2 (2003): 120-139.
[13] Humphreys, Greg, et al. "Chromium: a stream-processing framework for interactive rendering on clusters." ACM Transactions on Graphics (TOG). Vol. 21. No. 3. ACM, 2002.
[14] Babcock, Brian, et al. "Models and issues in data stream systems." Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, 2002.
[15] Grell, Stephan, and Olivier Nano. "Experimenting with complex event processing for large scale Internet services monitoring." 1st International workshop on Complex Event Processing for the Future. 2008.
[16] Wang, Yongheng. "A Proactive Complex Event Processing Method for Intelligent Transportation Systems." Lecture Notes on Information Theory Vol 1.3 (2013).