SERBIAN JOURNAL OF ELECTRICAL ENGINEERING
Vol. 12, No. 1, February 2015, 117-127

UDC: 004.422.635.5 **DOI: 10.2298/SJEE1501117J**

Data Stream Management System for Moving Sensor Object Data

Željko Jovanović[1]

Abstract: Sensor and communication development has led to the development of new types of applications. Classic database data storage becomes inadequate when data streams arrive from multiple sensors. Then, data querying and result presentation are not efficient. The desired results are obtained with a delay, and the database is filled with a large amount of unnecessary data. To adequately support the above applications, Data Stream Management System (DSMS) applications are needed. DSMSs provide real-time data stream processing. In this paper, a client-server system is presented with DSMS realized on the Java WebDSMS application server side. WebDSMS functionalities are tested with simulated data and in real-life usage.

Keywords: CEP, DBMS, DSMS, Data Streams, Simulation.

1 Introduction

With the development of rapid sensor technology, wireless communications, and mobile positioning technology, there has been a significant increase in the amount of data to be acquired, processed, and visualized. Data from sensors come in the form of continuous Data Streams (DSs). It is necessary to obtain useful information for system users from incoming data streams in real time without client activity. Applications that deal with continuous data streams are becoming increasingly important in environmental monitoring, surveillance, tracking, telecommunications, and plant maintenance. These application functionalities need to be provided with Data Stream Management System (DSMS) support. Moreover, in many of these applications, there is a need to ask questions that require stored historical data to be compared and combined with real-time streaming data, which makes decision-making very complex. This is called Complex Event Processing (CEP). Existing database management systems (DBMSs) store data for later querying according to the client's activity. Unlike DBMS, DSMS analyses incoming data arriving as continuous streams of data in real-time to select data of interest for permanent storage in a database and live data presentation to a client. In DBMS, data are passive and clients are

[1]Faculty of Technical Sciences Čacak, University of Kragujevac, Svetog Save 65, 32000 Čačak, Serbia;
E-mail: zeljko.jovanovic@ftn.kg.ac.rs

active, while in DSMS, data are active while clients are passive. This means that without client activity, only important data will be presented. For widespread usage, DSMS applications should be Web based. Since DSMSs have to analyse large amounts of data with great accuracy, it is necessary to test their functionalities before real-world usage. In this paper, one Web-based DSMS (WebDSMS) is tested with a Web-based data stream simulator (DSSimulator). DSSimulator produces controlled data so the functionality of WebDSMS can be verified. The amount of data and number of simulated clients can be changed in order to test WebDSMS in different scenarios.

2 Existing DSMSs

There are several DSMS projects which offer different functionalities: the Aurora, Borealis, STREAM, Cougar, TelegraphCQ, Infosphere streams, Microsoft StreamInsight, and Esper projects.

Aurora [1] was the first DSMS and was developed through the cooperation of Brown, Brandis, and MIT University. The primary goal of the Aurora project was to build a single infrastructure that could efficiently and seamlessly meet the requirements of applications. It addresses three broad application types in a single, unique framework:

- *Real-time monitoring applications*,
- *Archival applications*,
- *Spanning applications*, which involve both present and past data.

It contains built-in support for eight primitive operations for expressing its stream processing. Among these are the Window, Slide, Tumble, Latch, and Resample operations. In 2005, Aurora was replaced by the Borealis project.

Borealis [2, 3] is a distributed multi-processor version of Aurora. Each Borealis node constitutes a single processing engine and executes a portion of the data flow network. Nodes execute a collection of operators called boxes. Because of its distributed architecture, a significant part of the project presents an algorithm for the distribution of jobs between nodes. Communication between Borealis nodes is carried out using XML. The network topology is dynamic, since XML-based messages can update the node catalogue to add or remove filter boxes. The Borealis project is no longer an active research project: the last release dates from summer 2008 and runs on Linux x86-based computers.

The STREAM project [4, 5], developed at Stanford University, attempts to provide complete DBMS functionality along with support for continuous queries over streaming data. The STREAM project reinvestigated data management and query processing in the presence of multiple, continuous, rapid, time-varying data streams. CQL (Continuous Query Language), which

implements the abstract semantics, is designed. The abstract semantics is based on two data types: streams and relations. Syntactically, CQL is a relatively minor extension to SQL. It offers Stream-to-Relation and Relation-to-Stream Operators. Since 2006, the STREAMS project is no longer in development.

The main idea of Cougar [6] is to work with small-scale sensors, actuators, and embedded systems and to transform them into a computing platform. Existing sensor networks assume that the sensors are preprogrammed and send data to a central node where the data are aggregated and stored for offline querying and analysis. Since the computing power of sensor nodes is growing, the Cougar project distributes queries to nodes and as a result only the desired data arrive in the central node. The biggest challenge is in query management and distribution through nodes. As a result, Sensoria Node and Mica Mote [7] were developed to execute queries over sensor nodes and to collect the desired measured parameters. The Cougar project is currently supported by two National Science Foundation Grants.

Telegraph [8] is a research project in UC Berkley's Computer Science Division. Its main goals are adaptive dataflow and querying streaming data from sensors. TelegraphCQ is the latest version of Telegraph. It allows querying of real-time and historical data thanks to parallelized operators. Offers querying flexibility based on the periodic reports by the CQL query language. Continuous data are indexed with the FastBit index system with the SQL interface. It works with an append-only dataset and as a result compressed bitmap indices are created for faster querying.

IBM® InfoSphere® [9] Streams is an advanced analytic platform that allows user-developed applications to quickly, analyse and correlate information as it arrives from thousands of real-time sources. A key differentiator for InfoSphere Streams is its distributed runtime platform, clustered for near limitless capability. It can be scaled from a single server to an unlimited number of nodes to process millions of events per second with microsecond latency. It offers geospatial querying.

Microsoft StreamInsight™ [10] is a powerful platform for developing and deploying CEP applications. It analyses and correlates data incrementally while the data are in-flight, without first storing them with very low latency. Developers can write their CEP applications using Microsoft's .NET languages such as Visual C#, leveraging the advanced language platform LINQ (Language Integrated Query) as an embedded query language. StreamInsight can be deployed in applications as a library or as a standalone server.

Esper [11, 12] is a component for CEP and event series analysis, available for Java as Esper and for .NET as NEsper. Esper and Event Processing Language (EPL) provide a highly scalable big data processing engine for historical data or live stream data. Esper is open source project and is provided

as a jar file. Functionalities defined in the jar file can be integrated in a Java application by importing the required libraries into the application's lib folder.

3 WebDSMS Java Web Application

As a demonstration, a WebDSMS Java Web application is developed with DSMS support. It is presented in Fig. 1 and developed in Model View Controller (MVC) architecture. The model consists of Java classes (Java bean) and database access object (DAO) classes. The data presentation is realized in the form of Java Server Pages (JSP) with Google Maps GIS support included.

The main part of WebDSMS is the controller part, which is realized as a Java servlet since it can be accessed through the URL address. The servlet URL address is an interface to WebDSMS functionalities to anyone around the world that provides data in the required data format.

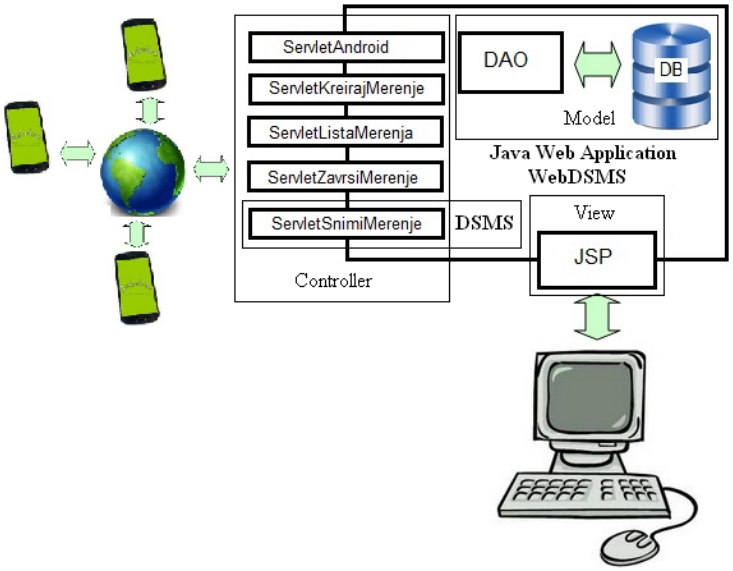

Fig. 1 – *WebDSMS architecture.*

This architecture is suitable for a client-server system with WebDSMS as a server part. The basic client-server system is presented in [13]. There are two types of clients (Web clients and sensor data sources, i.e. smart phones). Since WebDSMS has servlet-based interfaces, sensor data sources need to provide the required URL request format. For this purpose an Android client application is developed. It use five servlets, shown in Fig. 1, in WebDSMS as interfaces with server part functionalities. As a demonstration of client-server communication URL format, the request and response data format for *ServletSnimiMerenje* is

presented in Fig. 2. The request URL format is shown at the top of Fig. 2. It consists of idK (client identification), opis (textual description or sensor values), longitude, and latitude. Accepted parameter values are analysed by DSMS functionalities.

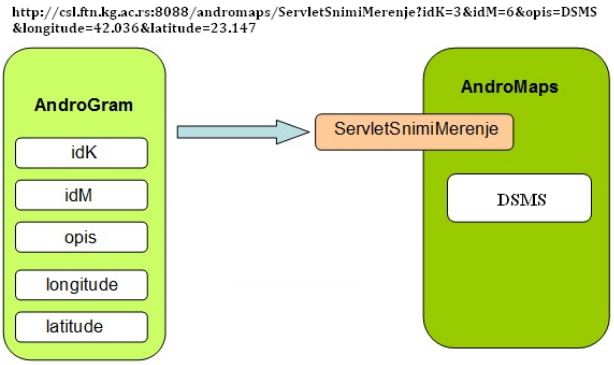

Fig. 2 – *ServletSnimiMerenje interaction.*

In the basic client-system version [13], all arriving data (important and not important) are stored in the MySql database for client querying. Data on client activity are presented without real-time event detection. For real-time event detection, DSMS functionalities are realized in *ServletSnimiMerenje.* Real-time event detection is done by real-time data querying. Only events of interest (data) are stored in the database and presented by Google Maps. The DSMS algorithm is presented on Fig. 3. The WebDSMS functionalities depend on user interests.

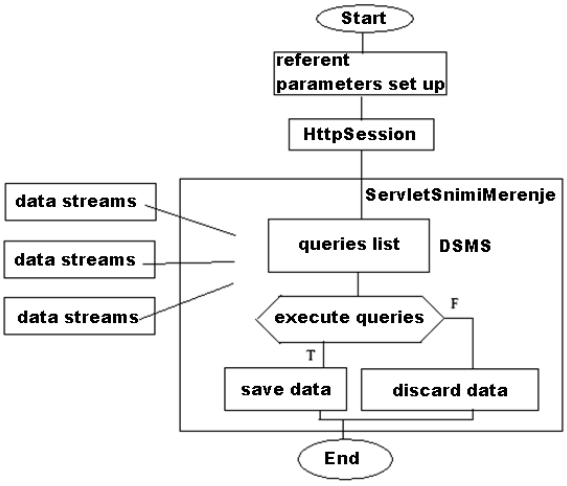

Fig. 3 – *DSMS algorithm.*

User set up referent parameter values through user interface for desired complex events real time monitoring. Set up parameters are user personalized (each user adjust its on parameters) and need to be accessible from *ServletSnimiMerenje.* That is why parameters are saved in HttpSession attribute *userParameters*. The WebDSMS is realized to accept correctly formatted requests from any sensors client. Sensors client produce data streams with periodic sensor sampling values. Data streams are processed with queries set up by the WebDSMS client according to the user interest. For demonstration these functionalities are realized:

- *boolean* ***manjeRastojanje****(retLat, refLon, lat, lon, refDistance)* – check distance between arriving coordinates (lat, lon) and referent coordinates (refLat, refLon) set up by the WebDSMS client is lower than refDistance.
- *boolean* ***vecaVrednost****(refVrednost, vrednost)* – check if value (*vrednost*) which arrived through URL request parameter "opis" (Fig. 2) is higher than referent threshold value (*refVrednost*), set up by the WebDSMS client.

In this WebDSMS version, client can set up to monitor sensors within set up distance or with sensor value higher than the *refVrednost* in real time. For this UserRefData Java bean class shown below is created.

```
public class UserRefData{
        private int userId;
        private double refLat;
        private double refLon;
        private double refDistance;
        private double refVrednost;
        private int query;                    // query type

        // constructor for manjeRastojanje query instance
        public UserRefData(int userId,
                        double refLat, double refLon, double refDistance)
{..}

        // constructor for vecaVrednost query instance
        public UserRefData(int userId, double refVrednost) {..}

}
```

This class has two types of constructors to create UserRefData class instance according to the client interests in real time monitoring. Next code shows adding UserRefData class instance to HttpSession object *userSession* as *userParameters* attribute of ArrayList<userRefData> type.

```
ArrayList<UserRefData> urdList = new ArrayList<UserRefData>();
UserRefData urd = new UserRefData(3, 30);
urdList.add(urd);
HttpSession userSession = request.getSession();
userSession.setAttribute("userParameters", urdList);
```

In *ServletSnimiMerenje s*et up queries from HttpSession attribute *userParameters* are executed over arriving data. The data with true queries return value are stored to database with information about query (query = 1 or query = 2) which returned true result. This information is necessary for real time sensors display in *manjeRastojanje query* satellite view (Fig. 4a.) or *vecaVrednost query map view* (Fig. 4b.). The false query return data are rejected. Client can turn queries on and off and change query parameters values in real time. If all queries are turned off (*urdList* size = 0) than all arriving sensor data will be rejected.

Main problem for every DSMS is higher calculation over arriving data. If the sensor client nodes are able to do the processing (smart phones), data processing can be distributed to them. Only desired values would arrive to WebDSMS which would reduce server data processing. This is possible if the correlation between deferent sensor nodes in CEP is not necessary.

This WebDSMS realization is possible to upgrade according to user interests. New methods (functionalities) can be added. As a result more CEP could be done and more complex events could be detected.

4 WebDSMS usage

The realized client-server system has a wide range of uses with functionality of storing relevant data on a database and real-time events presentation to clients. In this paper, real and simulated uses are presented.

As real usage, Android clients are set up to send their location and moving speed every second. In WebDSMS, *manjeRastojanje* query is turned on to detect clients' events with a distance of less than 5 km from Cacak city centre. Real-time presentation is shown in Fig. 4a. The *vecaVrednost* query is set up to detect clients' events with *opis* parameter values higher than 30. The real-time presentation is shown in Fig. 4b.

As simulated usage, DSSimulator is developed. It can generate a large amount of the data in a controlled scenario, which is important for testing DSMS functionalities. If collection of the data which arrive at the DSMS system is known, the result can be calculated in advance, and the result can be compared with the result produced by simulation for validation of the functionalities of DSMS. DSSimulator is developed as a Java Web application whose architecture and algorithm are presented in Fig. 5.

(a) (b)

Fig. 4 – *Query results data presentation:* (a) *manjeRastojanje query satellite view* (b) *vecaVrednost query map view.*

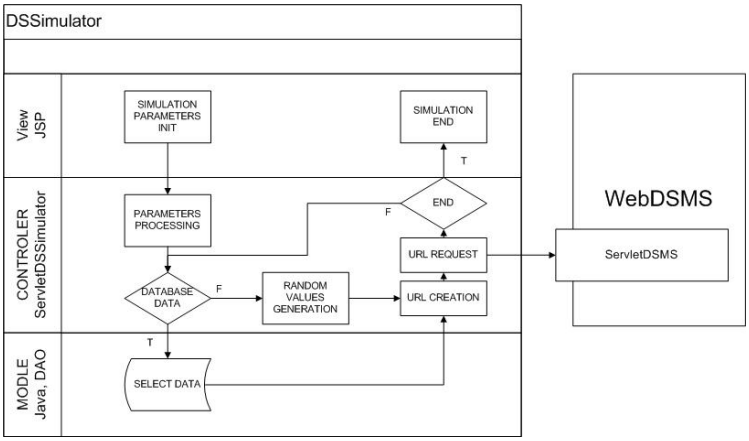

Fig. 5 – *DSSimulator architecture and algorithm.*

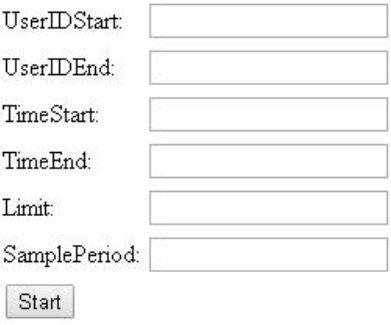

Fig. 6 – *DSSimulator parameter initialization form.*

DSSimulator is developed in Java using MVC architecture. From the Web browser, the client sets up the initialization parameters for simulation. The main functionalities are achieved by the controller part of the application in ServletDSSimulator. According to client demand, simulation data can be selected from a database or randomly generated. One data row contains the userid, timestamp, longitude, latitude, and speed. After that, the URL request is created and sent to the predefined URL address of ServletSnimiMerenje in the WebDSMS application. The entire process is repeated until the simulation ends. DSSimulator initialization parameters are presented in Fig. 6.

The client can set up idK range values (*UserIdStart, UserIdEnd*), time start and end values (*TimeStart, TimeEnd*), a data limit per client (*Limit*), and a sleep interval time between streams (*SamplePeriod*). For the data accuracy monitoring, a range of 200 users is set with 10 iterations and a location limit of 1000 location data. Each iteration is followed by a sleep period of 10 seconds for reviewing the detection of events. This means that DSSimulator sends up to 1000 requests per second to WebDSMS and then waits 10 seconds. This sequence is repeated until the end-time parameter is reached. Since a large amount of data arrive at WebDSMS, every request is logged in the server and shown in the console by its ID. Requests created by DSSimulator are logged as "DSSimulator LOG: *request id*" and requests received at WebDSMS are logged as "WebDSMS LOG: *location parameters*". A detected event that satisfies the statement conditions is also logged on the server and console in full data format as "Event detected: *event*". Part of the logged data with detected events are shown in Fig. 7.

```
DSSimulator LOG: 77
DSMS log - 77 - 0 - 2014-08-17 08:00:00.0 - 336 - 318 - 45.494783268 - 9.131159848 - 0
DSSimulator LOG: 78
DSMS log - 78 - 0 - 2014-08-17 08:00:00.0 - 265 - 287 - 45.499881962 - 9.11486052 - 0
DSSimulator LOG: 79
DSMS log - 79 - 0 - 2014-08-17 08:00:00.0 - 394 - 583 - 45.451197658 - 9.144474792 - 0
DSSimulator LOG: 80
DSMS log - 80 - 0 - 2014-08-17 08:00:00.0 - 458 - 800 - 45.4155068 - 9.159167144 - 0
DSSimulator LOG: 81
DSMS log - 81 - 0 - 2014-08-17 08:00:00.0 - 140 - 489 - 45.466658214 - 9.08616452 - 0
DSSimulator LOG: 82
DSMS log - 82 - 0 - 2014-08-17 08:00:00.0 - 417 - 564 - 45.454322664 - 9.149754856 - 99
-------------------------------------------------
        Event 2 detected: 82 - 0 - 2014-08-17 08:00:00.0 - 417 - 564 - 45.454322664 - 9.149754856 - 99
-------------------------------------------------
82 - 0 - 2014-08-17 08:00:00.0 - 417 - 564 - 45.454322664 - 9.149754856 - 99
VELICINA LISTE :4
DSSimulator LOG: 83
DSMS log - 83 - 0 - 2014-08-17 08:00:00.0 - 607 - 356 - 45.488533256 - 9.193372776 - 0
DSSimulator LOG: 84
DSMS log - 84 - 0 - 2014-08-17 08:00:00.0 - 734 - 470 - 45.46978322 - 9.222527912 - 0
DSSimulator LOG: 85
DSMS log - 85 - 0 - 2014-08-17 08:00:00.0 - 480 - 266 - 45.503335916 - 9.16421764 - 114
-------------------------------------------------
        Event 2 detected: 85 - 0 - 2014-08-17 08:00:00.0 - 480 - 266 - 45.503335916 - 9.16421764 - 114
-------------------------------------------------
85 - 0 - 2014-08-17 08:00:00.0 - 480 - 266 - 45.503335916 - 9.16421764 - 114
VELICINA LISTE :5
DSSimulator LOG: 86
DSMS log - 86 - 0 - 2014-08-17 08:00:00.0 - 305 - 214 - 45.511888564 - 9.12404324 - 0
DSSimulator LOG: 87
DSMS log - 87 - 0 - 2014-08-17 08:00:00.0 - 186 - 529 - 45.460079254 - 9.096724648 - 0
DSSimulator LOG: 88
DSMS log - 88 - 0 - 2014-08-17 08:00:00.0 - 528 - 666 - 45.437546316 - 9.175236904 - 0
DSSimulator LOG: 89
```

Fig. 7 – *Logged data, detected event.*

As a result, only detected events are inserted into the database and data of no interest are discarded. Inside WebDSMS, two counters are implemented to count the number of arriving events and events of interest. In this demonstration, only 2% of data are events of interest. This produces a memory saving of 98% but increases the processing hardware dependencies. Regarding this, the greatest benefit is live event detection. Google Maps content changes without client activity, providing efficient functionalities in live visual monitoring.

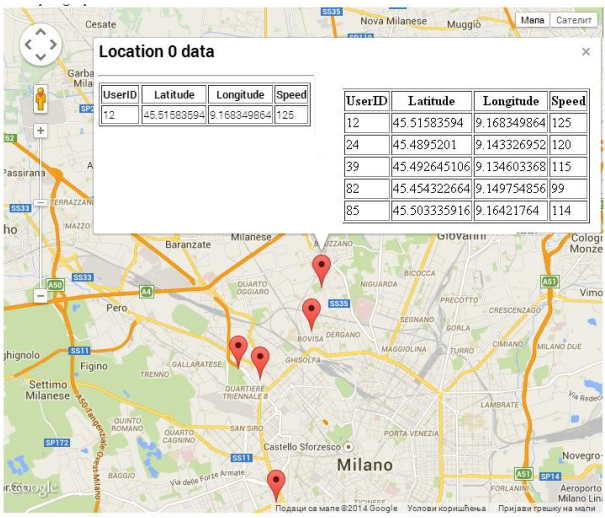

Fig. 8 – *GIS presentation of detected vecaVrednost events.*

Events detected for a *vecaVrednost(50)* query are presented in Fig. 8. The client can see all the event data by clicking on a marker. By analysing the logged data, WebDSMS functionalities are proven.

5 Conclusion

DSMS applications provide advanced functionalities. Real-time CEP is most important. Since DSMSs analyse a large amount of data, it is necessary to test the results they provide. After successful testing, WebDSMS can be used in the real world for analysing a large amount of data arriving from sensor network nodes. Since it is Web-based, every device with Internet connectivity can produce data for stream analysis if it creates a valid URL request. Smart phones can be used as data providers and complex events can be detected.

Acknowledgements

The work presented in this paper was funded by grant no. TR32043 for the period 2011–2014 from the Ministry of Education and Science of the Republic of Serbia.

6 References

[1] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seideman, M. Stonebraker, N. Tatbul, S. Zdonik: Monitoring Strams – A New Class of Data Management Applications, Brown University, Aurora project, http://cs.brown.edu/research/aurora/vldb02.pdf .

[2] J. D. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik: The Design of the Borealis Stream Processing Engine, 2nd Biennial Conference on Innovative Data Systems Research (CIDR'05), Asilomar, CA, USA, January 2005.

[3] http://cs.brown.edu/research/borealis/public/

[4] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, J. Widom: STREAM: The Stanford Data Stream Management System, http://ilpubs.stanford.edu:8090/641/1/2004-20.pdf

[5] S. Babu, J. Widom: Continuous Queries over Data Streams, SIGMOD Record, Vol. 30, No.3, 2001, pp. 109 – 120.

[6] N. Trigoni, Y. Yao, A. J. Demers, J. Gehrke, R. Rajaraman: Multi-query Optimization for Sensor Networks. International Conference on Distributed Computing in Sensor Systems. DCOSS'2005, Heidelberg, Germany, 2005, pp. 307 – 321.

[7] S. Madden, J. Gehrke: Query Processing in Sensor Networks, IEEE Pervasive Computing, Vol. 3, No. 1, 2004, pp.46 – 55.

[8] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, M. A. Shah: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World, CIDR 2003.

[9] http://www-03.ibm.com/software/products/en/infosphere-streams

[10] http://technet.microsoft.com/en-us/library/ee362541.aspx

[11] T. Bernhardt, A. Vasseur, A. Esper: Event Stream Processing and Correlation, O'Reilly Media, August 2007.

[12] http://esper.codehaus.org/

[13] Z. Jovanovic, N. Pantelic, S. Starcevic, S. Randjic: Web Gis Platform with Android Mobile Gis Client, International Scientific Conference UNITECH 2013, 22–23 November 2013, Gabrovo, Bulgaria, Vol. II, pp. II.113 –II.116.