Expert Systems with Applications 38 (2011) 6530–6539

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Event-driven architecture for decision support in traffic management systems

Jürgen Dunkel [a,*], Alberto Fernández [b], Rubén Ortiz [b], Sascha Ossowski [b,*]

[a] Hannover University of Applied Sciences and Arts, Ricklinger Stadtweg 118, 30459 Hannover, Germany
[b] University Rey Juan Carlos, C/Tulipán s/n, 28933 Móstoles, Madrid, Spain

ARTICLE INFO

Keywords:
Event models
Event-driven architecture
Complex event processing

ABSTRACT

Decision support systems for traffic management systems have to cope with a high volume of events continuously generated by sensors. Conventional software architectures do not explicitly target the efficient processing of continuous event streams. Recently, event-driven architectures (EDA) have been proposed as a new paradigm for event-based applications. In this paper we propose a reference architecture for event-driven traffic management systems, which enables the analysis and processing of complex event streams in real-time and is therefore well-suited for decision support in sensor-based traffic control systems. We will illustrate our approach in the domain of road traffic management. In particular, we will report on the redesign of an intelligent transportation management system (ITMS) prototype for the high-capacity road network in Bilbao, Spain.

1. Introduction

Decision support systems (DSS) are information systems that provide assistance to humans involved in complex decision-making processes. The fundamental task for modern DSS is to help decision-makers in building up and exploring the implications of their judgements (French, 2000). DSS are indispensable components of intelligent traffic management systems (Ossowski et al., 2005) because operators must come to decisions in a short reaction time based on a huge amount of traffic data.

Nowadays, most traffic control systems are sensor-based: loop-detectors installed in the roads emit events when cars are passing (Cuena, Hernández, & Molina, 1996). Furthermore, GPS devices or vehicle information and communication system (VICS) are sending continuously vehicle positioning data (Inoue, Shozaki, & Kakuda, 2007). These technologies provide a very high volume of fine-grained individual events which must be processed to analyze the current traffic situation and to take appropriate control actions.

A key issue of decision support systems in traffic management is short latency. The gap between the time the sensor data is emitted and the time the analysis results are available should be as short as possible, e.g. to cope with a traffic accident. Appropriate systems must provide real-time decision support (Delic, Douillet, & Dayal, 2001) or "Zero Latency" responsiveness as proposed by GARTNET-GROUP (2007).

Traffic control systems must analyze millions of events continuously produced by sensors and vehicles to detect problems immediately and to trigger suitable control mechanism.

Current software architectures of decision support systems are not targeted on those sensor-based systems, i.e. they cannot deal with the efficient processing of continuous event streams. Conventional service-oriented architectures (SOA) are based on a process-oriented control flow, which is not appropriate to event-driven systems. Due to the high volume of events and their complex dependencies it is not possible to specify a predefined process flow. Other approaches, such as mainstream, AI-based multiagent architectures (Bandini, Bogni, & Manzoni, 2002; Dresner & Stone, 2005; Ossowski, Hernández, Iglesias, & Fernández, 2002; Roozemond, 1999) focus on knowledge processing, but do not explicitly target the problems associated to real-time high-volume event processing. Furthermore, they lack inherent concepts for describing temporal dependencies between events, which are crucial to react appropriately in certain situations (Lamport, 1978).

In recent years, event-driven architectures (EDA) have been proposed as a new architectural paradigm for event-based applications (Luckham, 2002). The main idea lies in the processing of events as the central architectural concept. In this paper, we propose a reference architecture for event-driven decision support systems. The key concept of our approach is to use complex event processing (CEP) as the process model for event-driven decision support. Event streams generated by sensors and vehicles contain a large volume of different events, which must be transformed, classified, aggregated and evaluated to initiate appropriate actions. Our reference architecture enables the analysis and processing of complex event streams in real-time and is therefore well-suited for decision support in sensor-based traffic control systems.

* Corresponding authors. Tel.: +49 511 9296 1823; fax: +49 511 9296 1810 (J. Dunkel), tel.: +34 916647485; fax: +34 914888530 (S. Ossowski).

E-mail addresses: juergen.dunkel@fh-hannover.de (J. Dunkel), alberto.fernandez@urjc.es (A. Fernández), ruben.ortiz@urjc.es (R. Ortiz), sascha.ossowski@urjc.es (S. Ossowski).

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.11.087

The remainder of the paper is organized as follows. In the next section, we present a distributed EDA-based reference architecture for decision support in traffic management systems. Section 3 illustrates the effectiveness of our approach through the redesign of an intelligent transportation management system (ITMS) for managing the high-capacity road network of the Basque town of Bilbao. Section 4 presents the results of the abstract architecture presented in this paper applied in the eDraft project (eDraft project, 2009). Finally, we summarize the most significant features of our approach and provide a brief outlook on future lines of research.

2. EDA for transport control systems

2.1. Event-driven architecture – Overview

Event-driven architecture (EDA) provides an architectural concept for dealing with complex event streams using *Complex Event Processing* (CEP) as event processing model (Luckham, 2002). Because sensor-based traffic management systems are emitting continuously data they are particularly suited for CEP.

One main principle of CEP is that events are not independent from each other, but correlated. Sensor data tends to be strongly correlated in both time and space. For instance, traffic data measured at one sensor is highly correlated to the data at an adjacent sensor. Similarly, readings observed at one time instant are highly indicative of the readings observed at the next time instant. Applications are not interested in individual readings in time or individual devices in space, but rather in application-level concepts of temporal and spatial granules (Jeffery, Alonso, Franklin, Hong, & Widom, 2006).

The main task of CEP is *Event Pattern Matching* to identify in a huge event cloud those patterns of events which are significant for a business domain. In traffic control systems millions of sensor-emitted events are analyzed to discover event patterns signifying upcoming traffic problems. Pattern matching and event processing are performed by so-called event-processing agents (EPAs), which monitor the event streams. Essentially, EPAs filter, split, aggregate, transform and enrich events. Furthermore, they synthesize new complex events from simple events. This step takes correlations between events into account and provides the real power of complex event processing.

Fig. 1 shows the building blocks of a Complex Event Processing (CEP) component. To automate event processing, a formalism based on metadata is required. The *Event Model* precisely defines the set of possible events with their constraints and interdependencies. Each event contains general metadata (event ID, event type, event timestamp) and event-specific information, e.g. the ID of a loop detector or a car ID. *Event Processing Rules* define correlations between events in form of event patterns, and determine corresponding actions. They can be expressed by event processing languages (EPLs) based on event algebras (Paton, 1999; Schiefer, Rozsnyai, Rauscher, & Saurer, 2007) or as SQL-like queries over event streams (Arasu, Babu, & Widom, 2003). Event processing is based on the *Event Data*, which manifests the occurred events, i.e. a set of instances related to event types specified in the event model. An *Event Processing Engine* is a rule engine executing event processing rules on the event data.

Due to the continuously arriving data, the event processing engine has to cope with infinite data streams. Therefore, the pattern matching is characterized by *continuous queries* that are issued once and then run continuously over the data stream (Babu & Widom, 2001). To cope with the infinite number of data, *sliding windows* are used, i.e. a historical snapshot considering the most recent set of events. It means that events have a certain lease time and are deleted if it is expired. The appropriate lease times are event-type specific: the more abstract an event is, the longer its lease time is.

2.2. EDA-architecture for decision support systems

The key issue of EDA-based systems is a precise event model, which reflects the different stages of event processing and yields the basis for the overall system architecture. Generally, all events can be structured in a layered hierarchy: for decision support systems based on sensor data we can derive a course-grained event hierarchy, as depicted in Fig. 2.

- *Raw Sensor Events*: are the physical events emitted by the sensors, e.g. loop detectors. Due to technical problems sensor data is often inconsistent: missed and duplicated readings, as well as unreliable readings causing outliers must be compensated (Bickel et al., 2007). Therefore, the raw sensor events are pre-processed and cleaned according to Jeffery et al. (2006) to overcome the inconsistencies.
- *Domain Data Events*: However, the cleaned sensor data is too fine-grained and still uncorrelated. DSS are not interested in individual sensors data, but in application-level concepts of spatial and temporal granules. Therefore, domain data events are derived by mapping raw sensor event data to domain concepts (Dunkel & Bruns, 2008). For instance, domain data events correlate data of sensors located in one road segment, and evaluate average traffic data as traffic density and occupancy. Furthermore, regular traffic behavior can be calculated by analyzing historical sensor data.
- *Problem Events*: In a problem identification step the domain

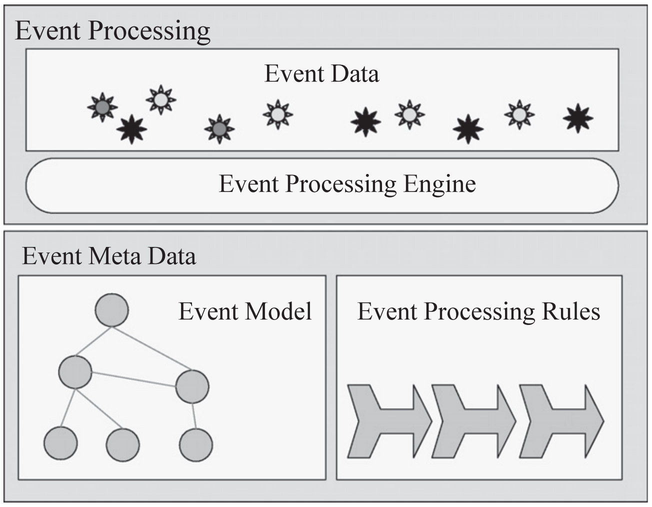

Fig. 1. Complex event processing components.

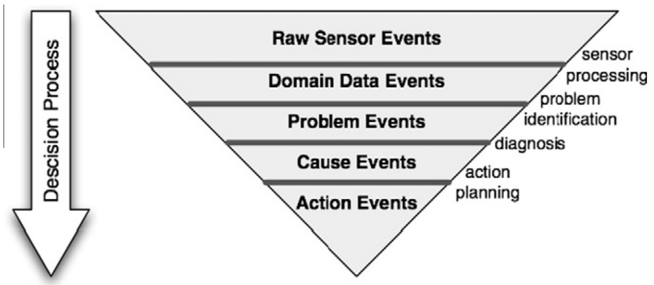

Fig. 2. Event hierarchy in decision support systems.

data events are synthesized to problem events, which characterize a situation that is undesirable for the decision-maker. For example, a problem event can identify high traffic density, traffic congestion or a blocked road.
- *Cause Events*: In a diagnosis step the problem events must be transformed into a number of cause events, i.e. an explanation in terms of causal features. A cause event hierarchy classifies the possible reasons for a problem. For example, traffic congestions in a certain area can be caused by an evening rush hour or by an accident in an adjacent area.
- *Action Events*: Based on the diagnosed cause events, an action planning step is needed, which yields a sequence of actions to eliminate the related causes and damages. Traffic rerouting to bypass a blocked road is an example of a possible action. In EDA actions are mapped onto special action events, which characterize the functional output of the DSS.

The event hierarchy reflects the decision process, which can be understood as a sequence of event processing steps. The system transforms the raw sensor events into more abstract and sophisticated domain events for evaluating the actual traffic situation and initiating appropriate traffic control steps.

Each of the event transformation steps is processed by corresponding event processing agents (EPA), which compose an event processing network (EPN) (Sharon & Etizon, 2008). Accordingly, we can derive a reference architecture as shown in Fig. 3. Each event processing agent represents an expert in the decision process.

The Sensor Processing Agent (SPA) provides the data cleaning of the raw sensor events and maps them subsequently to the domain data events. Domain data events are expressed in domain concepts, as road segments and traffic data.

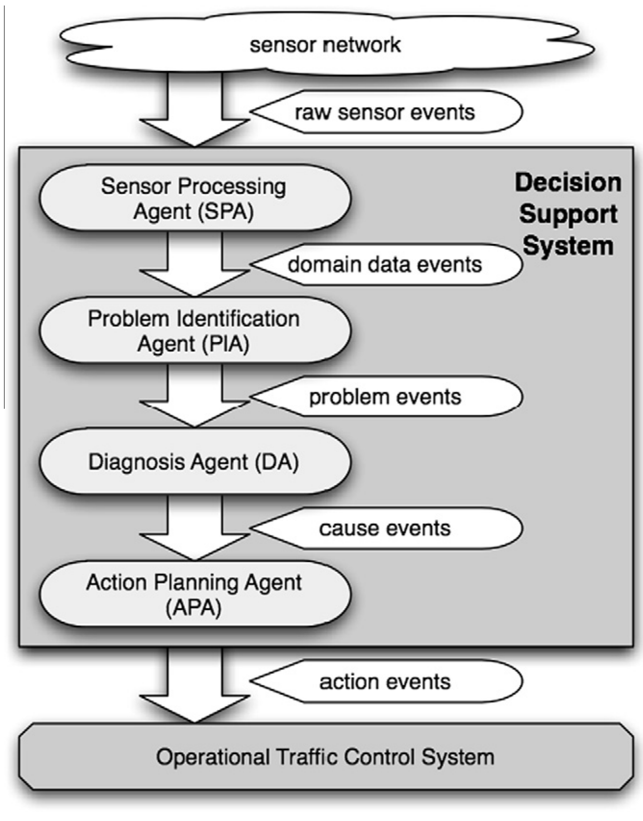

Fig. 3. EDA-based reference architecture for DSS.

The Problem Identification Agent (PIA) executes continuous queries on the stream of domain data events to derive traffic problems. A traffic problem can be understood as a state which deviates from normal traffic conditions. For example, congestions are characterized by the fact that the traffic demand exceeds the roads capacities.

The Diagnosis Agent (DA) analyzes the incoming problem events and applies appropriate pattern to create complex cause events. The event pattern rules infer from the incoming problem events a set of possible causes.

The Action Planning Agent (APA) executes the final decision process step. It generates appropriate action events by processing the cause events. The sequence of generated action events are resulting from an action planning task. Ideally, the executed action plan should yield to a situation where most damages have been alleviated and all the problem causes have been eliminated. The general reasoning method to deal with this functionality is a planner, using classification reasoning on predefined plans and subplans.

2.3. Distributed architecture

Real traffic scenarios exhibit a distributed structure: sensors are dispersed all over the topology, and traffic flows can often be consistently conceived within local areas of the network. Because a well-adapted software architecture should reflect this *a priori* structure, we transformed the proposed reference architecture into a distributed design, according to Fig. 4.

In each road section a set of sensors are situated, whose events are captured and transferred into domain concepts by a dedicated Sensor Processing Agents (SPA). These sections define *proximity groups* (Jeffery et al., 2006) related to a set of sensors monitoring the same spatial granule. The next steps are defined by the decision process and performed by a set of Problem Identification Agents (PIA), Diagnosis Agents (DA) and Action Planning Agents (APA). Each of these agents makes its own local decisions based on its upstream events. The agents use their individual event processing rules modelling the local circumstances.

Finally, there is a central Coordination Agent, which collects the action events of the APAs and resolves possible conflicts, e.g. if there are two contradictory proposals for traffic rerouting.

This distributed architecture is modularized and scalable. It reflects the spatial structure of sensor-based traffic control systems and the fact that decisions are usually made on base of local data and circumstances. The Coordination Agent solves conflicts between local decisions due to a global goals and strategies.

The communication between the agents is event-based as described in sub Section 2.2; standard message-oriented middleware (MOM) can be used as event channels to exchange data between the different event processing agents.

2.4. Tool support

Currently, a number of tools for developing EDA-based applications are available. Besides many research projects (Babcock, Babu, Datar, Motwani, & Widom, 2002; Wang & Liu, 2005; Wu, Diao, & Rizvi, 2006) there are already some commercial products available (Coral 8, 2008; IBM, 2009; Oracle, 2009; Streambase, 2009; Tibco BusinessEvents, 2009). Usually they provide an event processing language (EPL) and a corresponding processing engine. At the moment, no generally accepted standards exist for event definition, event pattern specification, or rule languages and engines. This lack of standardization is one of the major obstacles in order to fully exploit the benefits of event-driven architectures.

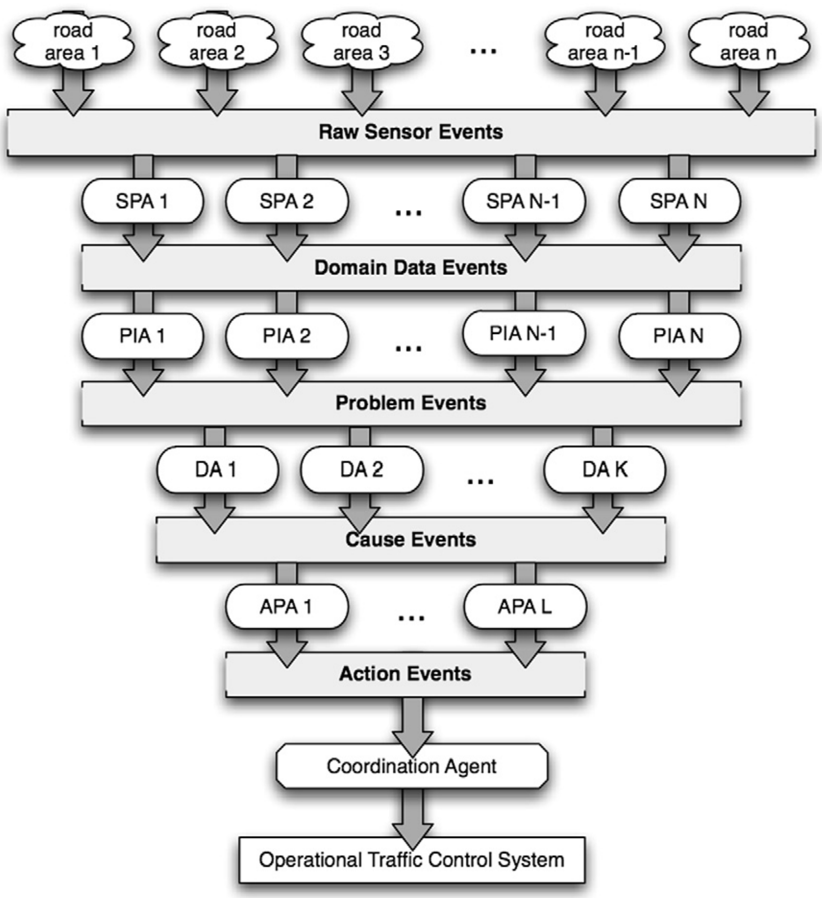

Fig. 4. Distributed DSS architecture.

EPLs can be expressed as ECA (event condition rules) (Paton, 1999; Zimmer & Unland, 1999) or as a SQL-based approach of continuous queries over event streams (Arasu et al., 2003). Recently, we have investigated the differences between the two approaches (Ermert, 2009) comparing the general rule system Jess (2009) with the Event Query Language of Esper, an open source event processing engine (ESPERTECH, 2008).

- General rule systems, like Jess (2009), ILOG JRules (2009) or JBoss Drools (2009) are not focused on event stream processing. They lack an inherent concept of time: Temporal dependencies between events and the handling of sliding windows must be integrated manually. Therefore, the description of event patterns is more complex and cumbersome. Furthermore, general rule systems are not optimized for dealing with a high volumes of continuously arriving events. Our experiments with Jess showed some significant performance drawbacks. For instance, only a few thousand events per second could be inserted into the Jess fact base. The Rete algorithm used in general rule-based systems does not support temporal operators for defining temporal constraints. Recently, some work has been proposed for integrating temporal constraints into Rete (Berstel, 2002; Walzer, Breddin, & Groch, 2008).
- By contrast, the concept of time is already integrated in Event Query Languages like ESPERTECH (2008), Coral 8 (2008), Streambase (2009). EQL-based systems explicitly target the efficient processing of complex event streams in real time. In our case study, experiments showed that Esper performs much better than Jess; more several 10.000 of events per second could be processed. Similar to databases, the response time depends mainly on the complexity of the event patterns, especially on how many event streams are joined.

Therefore, in the following case study we will use the open source event stream processing engine Esper, which provides its own continuous query language EQL (event processing query language) (ESPERTECH, 2008).

3. Case study

3.1. Scenario

We will illustrate our approach in the domain of road traffic management. In particular, we will report on the redesign, based on the EDA architecture put forward in the previous section, of a DSS prototype for managing a planned extension of the high-capacity road network in the greater Bilbao area in Spain (Ossowski et al., 2005).

Regular information about the traffic situation in this highly used area, registered by loop detectors, is received in the Mobility Management Centre. On the basis of this data, traffic operators have to take decisions on what control actions to take in order to solve or minimize congestion. These actions include:

(1) Displaying messages on Variable Message Signal (VMS) panels installed above the road to warn drivers about traffic problems or recommend alternative routes and/or.

(2) Contacting local authorities to send the right people to manage the situation.

The DSS has the purpose of assisting operators with their management task, helping them to configure consistent control plans for the whole road network, and making the best use of the available signal devices from a global perspective.

The structure of the high capacity road networks is modelled in terms of sections, whose shape and structure account for a certain capacity of traffic flow, connected either linearly or by ramps, as well as sensors (mostly loop detectors) and actuators (generally VMS) located at certain sections. In line with the aforementioned architecture, the conceptual vocabulary of our application is completed by a number of taxonomies, representing the set of possible sensor, domain data, problem, diagnosis and action events.

The reasoning steps of the system follow the same pattern:

- Firstly, rules listen to the continuous and massive stream of sensor events and, after performing correction, completion, etc. of sensor data, relate specific patterns of abstract data values (qualitative measures of speed, occupancy and flows) to problem types.
- There may be both generic rules, applicable to all type of networks with a certain topology (congestions due to excess demand on trunks), as well as specific patterns (applicable to a set of sections with a particular location in the network). Problems at certain sections, and their importance, are modelled in terms of traffic excess at those sections (comparing demand to capacity).
- Diagnosis is performed by determining the routes that drivers take from an origin *O* to a destination *D* that pass thorough a problematic section. The relevance of these causes is expressed in terms of the quantity of traffic flow that contributes to the detected excess, based on an estimation of the amount of flow between *O* and *D* for a specific time slot and day of the week, and of how much of this traffic will choose this route in the current context.
- Finally, action events are generated by rules that relate signal plans (coherent sets of messages on certain VMS) to a specific distribution of traffic from *O* to *D* among different routes.

It should be noticed that traffic engineers in the Mobility Management Centre usually conceive the road network in terms of so-called *problem areas* (PA) and *management areas* (MA). Problem identification and diagnosis on the one hand, and action planning on the other, can usually be modelled independently in each problem area, and each management area, respectively (see Fig. 5).

By consequence, at any of the different levels of our DSS architecture, there are several event processing agents of the same type (i.e. receiving the same type of events as input and producing the same type of event as output). In order to generate control action events, inputs from several different problem areas (i.e. their corresponding events) are usually necessary.

Furthermore, action events for different management areas may be in conflict: they may involve physical (e.g. setting different messages on the same VMS) or logical conflicts (e.g. alleviating a problem by deviating traffic to an already congested area). Context-sensitive priority rules assure that the set of action

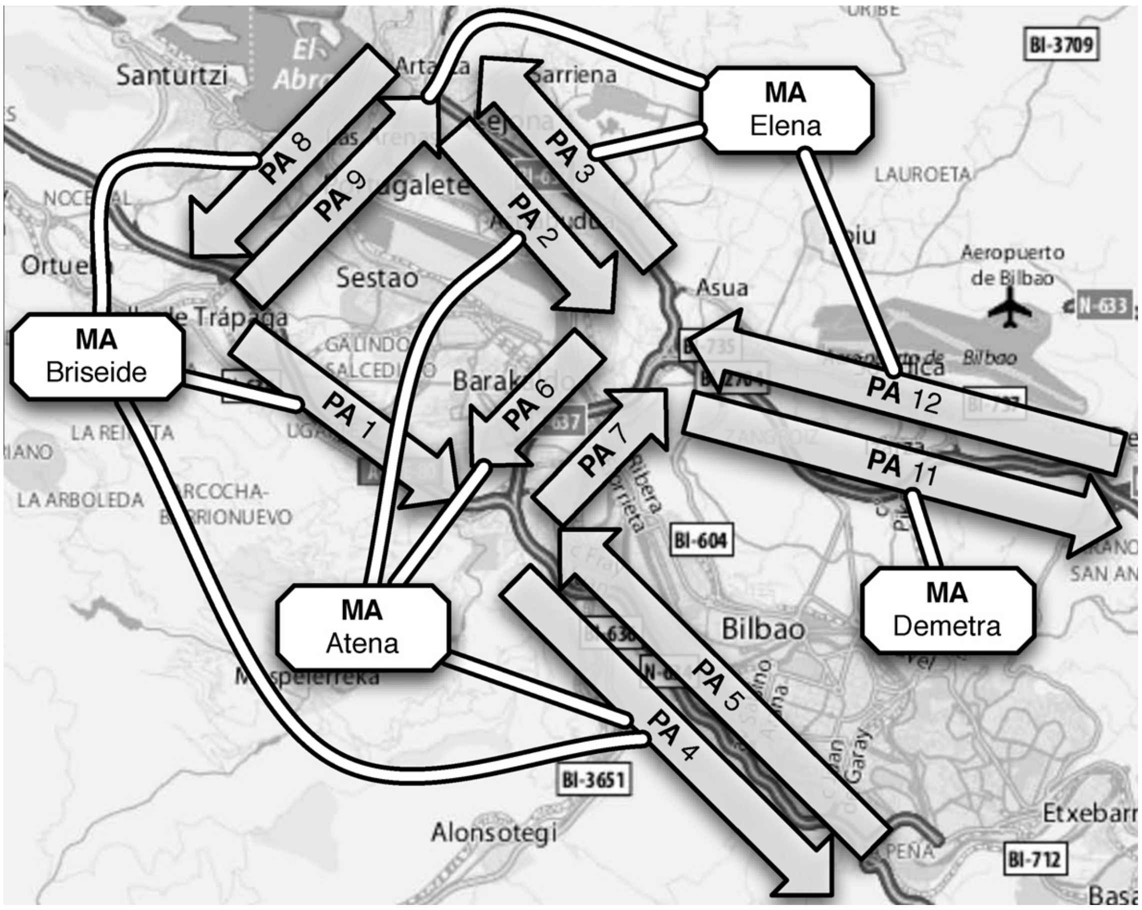

Fig. 5. Problem areas (PA) and management areas (MA).

events, which represent the recommendations of our systems to the operators in the Mobility Management Centre, are free of those conflicts.

3.2. Event model

The event flow in our application domain follows the event hierarchy for DSS described in Section 2.2 (Fig. 2). The following types of events can be identified.

Raw Sensor Events are the incoming events from the physical sensors located on the road. They contain event-specific information as a *sensor id* as well as quantitative measures of *speed*, *occupancy* and traffic *density*.

In our architecture *Domain Data Events* are split into different event types: First, events conveying information about sensors belonging to the same section of the road are aggregated into one *Section Event*, with the same attributes and completed with domain knowledge about its *capacity* and some information about the topology of the road (e.g. the next section in the traffic direction). Then, an abstraction process transforms the numeric values into discrete qualitative values generating *Section Abstraction Events*. For instance, the measured car velocities are related to the allowed speed limits and assigned to a qualitative category, e.g. 'SLOW', 'MEDIUM', 'FAST'.

With the symbolic data for each section, Problem Identification Agents (PIA) apply different traffic problem pattern to identify and generate *Problem Events*, which inform about the *location*, *type*, traffic *excess*, *state* and *category* of the problem.

Whenever a problem is detected, the Diagnosis Agent (DA) has to identify the cause of the problem, and *Cause Events* are generated containing information about the *problem type* and the *routes* contributing to it.

Finally, from information about problem causes, Action Planning Agents (APA) generate *Action Events* including information about action plan proposals, the location to execute the proposal action and an information message to specify additional information about the action.

3.3. Event processing rules

The following rules are examples of using the Esper engine and their own continuous query language EQL (ESPERTECH, 2008) to implement some event processing rules that the agents presented in the proposed architecture implement for the decision support process. The agents create these rules from their knowledge about the topology of their area of control and the expert knowledge about problem detection, diagnosis and control action planning.

3.3.1. Sensor processing agents (SPA)

In this section, we use the fragment of the topology of the high-capacity road in Bilbao shown in Fig. 6, which corresponds to a part of the logical problem area *Rontegi Sentido Aeropuerto/Avanzada*. In that figure, two sections and nine sensors are represented.

Domain Rules. The rule shown below contains the three main parts of EQL rules:

(i) *insert into* creates a new event of a certain event type (here: *SensorInSection*);
(ii) in the *select*-clause the fields of the new event are filled; and
(iii) source events are specified in the *from*-clause.

In particular, that rule adds topology information about sections in Bilbao's roads to the information obtained from all Sensor events generated by sensors located in the section '*Rontegi antes Erandio*', i.e. 302033, 302034, 302035 and 302036. The section name (id) is added to the sensor numerical data (*speed*, *occupancy*, *density* and *vehicles*) and completed with domain knowledge, like the *capacity* of the section and part of the topology such as *next_section* information.

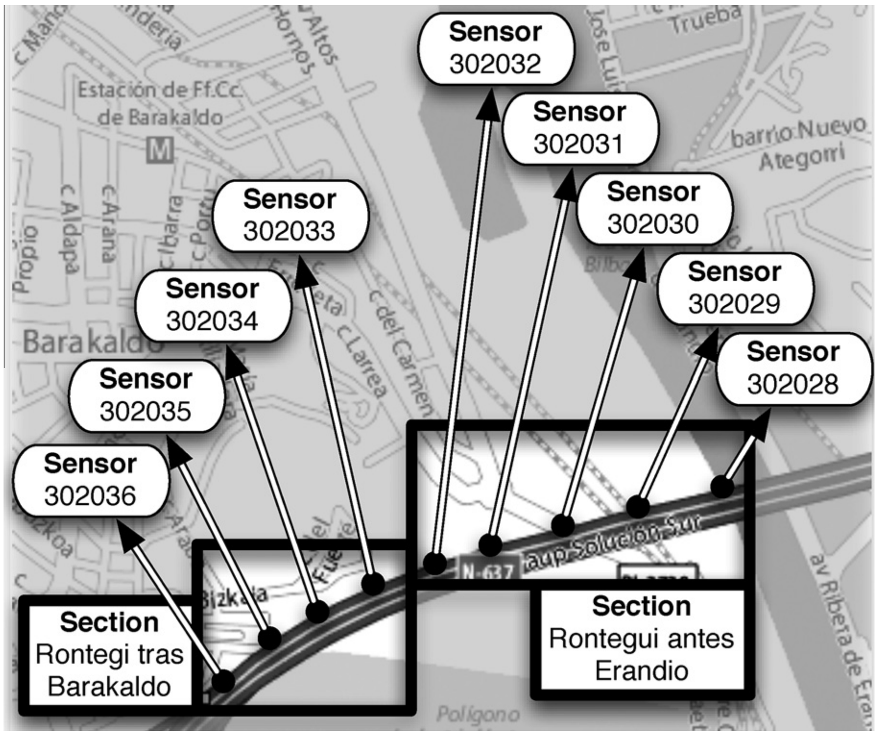

Fig. 6. Example of sensors positions in road sections.

There exist some similar rules which are only slightly different according to the structural properties of the road network. For instance, all sensor data belonging to the same road section should be aggregated. The following rule combines the sensor events of four adjacent sensors. A sliding window of length size 1 is used to achieve that each arriving *Sensor* event generates a new aggregated *SensorInSection* event. Such topology-dependent rules can be generated at compile-time from generic rule patterns based on topology knowledge, and are added automatically to the event-processing engine.

```
insert into   SensorInSection
select        'Rontegi tras Barakaldo' as id,
              2820 as capacity,
              'Rontegi antes Erandio' as next_section,
              speed as speed,
              occupancy as occupancy,
              density as density,
              vehicles as vehicles
from          Sensor(
              id='302033'
              or id = '302034'
              or id = '302035'
              or id = '302036').win:length (1);
```

Aggregation and Abstraction Rules. Traffic engineers use symbolic values (e.g. low speed, high density ...) rather than numerical in their knowledge representation of problem detection patterns. With this rule, numerical data provided by *SensorInSection* events are aggregated (all sensor in the same section, indicated in "group by") and abstracted into qualitative measures. In this example, we use an external java function to classify the speed, occupancy and density in three levels: *LOW*, *MEDIUM* and *HIGH*. The rule pattern uses a sliding window of 1 second, and generates a new *Section* event for each section every second (last line of the rule).

```
insert into  Section
select       id,
             capacity,
             next_section,
             sum (vehicles) as demand,
             Abstraction.absSpeed (avg (speed)) as speed,
             Abstraction.absOccupancy (avg (occupancy)) as
             occupancy,
             Abstraction.absDensity (avg (density)) as density
from         SensorInSection.win:time (1 seconds)
group by     id
output       last every 1 seconds
```

3.3.2. Problem identification agents (PIA)

Problem detection is based on a set of typical patterns that relate traffic conditions with problem types. In this section we present some problem patterns implemented as EQL rules used by the Problem Identification Agents.

Fig. 7 presents one of those patterns: 'retention caused by a trunk incident'. This pattern matches when a section is characterized by low speed or high occupancy and, in the subsequent section, shows low or medium density, and either high speed or low occupancy.

The next rule implements this pattern with *Bef* and *Aft* referring to two consecutive sections. The abovementioned conditions are represented in the *where*-clause of the EQL rule. When two specific consecutive sections fulfil the pattern, a problem is identified and the corresponding new *problem* event is generated. The problem event is completed with the specific characteristics of the identified problem (problem *state* and *category*).

```
insert into   Problem
select        Aft.id as location,
              'retention because of trunk incident'
              as description,
              'linear connection' as type,
              Aft.demand-Aft.capacity as excess,
              'incident' as state,
              'problem' as category
from          Section.win:time (30 seconds) Aft,
              Section.win:time (30 seconds) Bef
where         Bef.next_section = Aft.id
              and (Bef.speed = 'LOW'
              or Bef.occupancy = 'HIGH')
              and (Aft.density = 'LOW'
              or Aft.density = 'MEDIUM')
              and (Aft.occupancy = 'LOW'
              or Aft.speed = 'HIGH')
group by      Aft.id
output        last every 30 seconds;
```

Fig. 8 presents another problem pattern: 'traffic jam in road narrowing'. That pattern detects a congestion when the section containing the constriction shows low or medium speed and low or medium occupancy, while the subsequent section is characterised by a high density and medium or high speed.

The following rule, similar to the previous one, implements the generic pattern. *Bef* and *Aft* are two consecutives sections that match the pattern.

```
insert into  Problem
select       Aft.id as location,
             'traffic jam in road narrowing' as description,
             'narrow' as type,
             Aft.demand-Aft.capacity as excess,
             'free' as state,
             'incipient' as category
from         Section.win:time(30 seconds) Aft,
             Section.win:time(30 seconds) Bef
where        Bef.next_section = Aft.id
             and
             ((Bef.speed = 'LOW'
             or
             Bef.occupancy = 'MEDIUM')
             and (Aft.speed = 'HIGH'
             or Aft.occupancy = 'LOW')
             and (Aft.density = 'HIGH'))
             or
             (((Bef.speed = 'LOW'
             or
             Bef.occupancy = 'MEDIUM')
             and (Aft.density = 'HIGH')))
group by     Aft.id
output       last every 30 seconds;
```

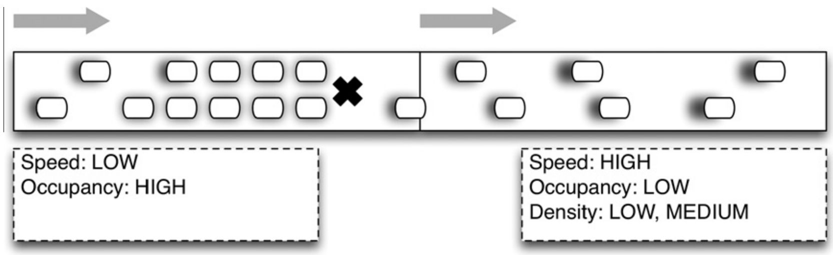

Fig. 7. Problem pattern 'retention because of trunk incident'.

3.3.3. Diagnosis agents (DA)

Most problems are mainly caused by a big amount of vehicles wishing to go through a given section. In our system the causes of a problem are given by the routes that "provide" vehicles to the congested section. The following rule is an example of such a rule, where the route *Rontegi sentido Aeropuerto/Avanzada* is considered as one of the causes for problems located in one of the four sections indicated in the *where*-clause of the rule. Cause events are generated every 60 seconds, so as to consider other co-occurring problems that might lead to a different cause inference.

insert into	Cause
select	'Rontegi sentido Aeropuerto/Avanzada' **as** route, category **as** problem_category, state **as** problem_state, location **as** problem_location
from	Problem.win:time(60 seconds)
where	location = 'Rontegi antes Barakaldo' **or** location = 'Rontegi tras Barakaldo' **or** location = 'Rontegi antes Erandio' **or** location = 'Rontegi tras Erandio'
group by	location
output	**last every** 60 seconds

3.3.4. Action planning agents (APA)

Several different actions can be proposed to solve existing problems The following two rules show examples of these types. Firstly, the next rule creates an action event, which specifies an alternative route if a cause event of the preceding rule has occurred.

insert into	Action
select	'alternative_route' **as** action, 'Rontegi sentido Aeropuerto/Avanzada' **as** location, 'Lamiako sentido Avanzada' **as** message
from	Cause.win:time(90 seconds)
where	route='Rontegi sentido Aeropuerto/Avanzada'
output	**last every** 90 seconds

The subsequent rule triggers an emergency call and an alert message in the appropriate message control panels as soon as a cause event with state 'incident' is observed. Occasionally, conflicting actions must be resolved by a Coordination Agent, e.g. contradictory proposals for traffic rerouting.

insert into	Action
select	'call_emergency_system' **as** action, problem_location **as** location, 'traffic incident' **as** message
from	Cause.win:time (90 seconds)
where	problem_state='incident'
output	**last every** 90 seconds

4. Results

In the eDraft project (event-driven architecture for traffic management) (eDraft project, 2009), we implemented the reference architecture presented in Fig. 3. Because in Esper the concept of processing agents is already integrated, it is straightforward to transfer our agent-based architecture to Esper: each processing agent can be mapped onto a dedicated instance of the Esper process engine. Distributing the event processing rules on different agents facilitates significantly the development of the traffic management system. Each agent contains only a small number of coherent processing rules making it easier to adjust, modify and debug them than in a single rule base.

Unfortunately, Esper does not provide any inherent concept for distributing events across multiple platforms. Thus, to realize our distributed architecture according to Fig. 4, inter-process communication mechanisms must be implemented manually. This could be achieved by communication proxies that forward events to Esper engines running on different machines using message-oriented middleware (MOM) or Java remote method invocation (RMI). In the eDraft project, we started with a single-server-based version of the decision support system, which we will migrate to a distributed architecture in the next future.

All the rules of the traffic management system have been specified using Espers Event Query Language (EQL) as shown in Section 3.3. Because of the build-in concepts of sliding windows and temporal constraints, it turned out that EQL is well-suited for defining event processing rules. Nevertheless, there is some conceptual weakness in Continuous Query Languages as we discussed in some more details in Dunkel, Fernández, Ortiz, and Ossowski (2009). For instance, in Esper no formal event model can be defined, but event definitions are hidden in low-level SQL-like code, i.e. event definition and event processing is intermingled.

In the eDraft project, we conducted two different experimental scenarios. First, we used the real-world scenario of the Bilbao traffic management system: we sent the traffic data measured in Bilbao as an event stream to our CEP component. The results of this experiment turned out satisfactory: the implemented rules were capable to detect the problems hidden in the real-world data; for instance, traffic jams or accidents.

Secondly, we developed a traffic simulator for creating a stream of raw sensor events which could be processed by our DSS. The simulator consists of the following building blocks:

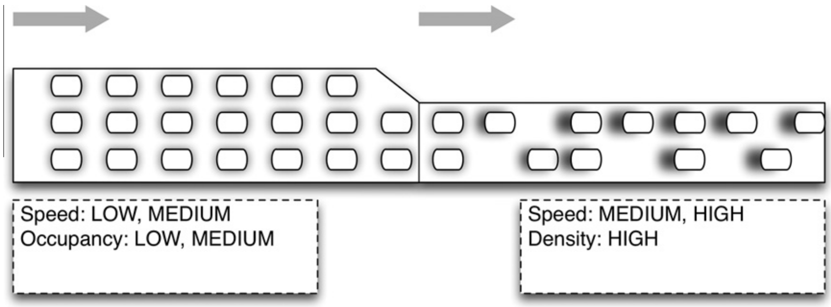

Fig. 8. Problem pattern 'traffic jam in road narrowings'.

- The simulated road area is specified in an appropriate XML format describing road parts, crossings and sensors.
- The system dynamics is based on a stochastic model, which specifies the arrival rate of cars, their velocities and their route. The simulator adapts the cars behavior to the current traffic situation; e.g. in case of a traffic jam they slow down.
- Finally, a graphical user interface visualizes the current traffic situation and allows to trigger traffic problems like accidents or slow cars. Vice versa, problems detected by the event processing agents of the DSS can be displayed.

The simulator allows us to prove our event processing rules and to adjust them. For instance, we learned that the length of sliding windows has a great impact on how fast traffic problems can be detected. Furthermore, the simulation experiments showed that event-driven architectures are well suited for decision support in sensor-based traffic control systems. Even high-volume event streams could be processed in real time.

5. Conclusions

In this paper, we have introduced an EDA-based reference architecture for DSS for traffic management. We have performed a re-design of an ITMS prototype for a real-world problem within the framework of that architecture, where event processing agents connect streams of increasingly abstract types of events, making use of a rule-based representation of local traffic expertise.

While mainstream, AI-based multiagent architectures are an adequate means of enacting structured knowledge models, they do not explicitly account for real-time sensor data processing. So, even though a set of implementation workarounds can assure an adequate response time for relatively small-scale traffic management DSS, the aforementioned fact is a major obstacle to the scalability of such systems. Furthermore, multiagent approaches lack inherent concepts of time: temporal constraints and sliding windows must be integrated manually.

On the other hand, event-driven architectures are known to be strong at processing high-volume complex event streams. In this case, we have shown that a real-world traffic management scenario can be effectively modelled and implemented within the EDA framework, thus greatly improving the real-time features of such systems and that this is possible without a need for complex structural transformation of knowledge models.

Based on a more complete and distributed implementation of the proposed architecture, we will carry out more quantitative evaluations of this approach. We expect further improvements in the response time, and the higher frequency of reasoning cycles, to lead to an enhanced performance of control proposals in certain situations. We also plan to look into other modern architectures from the field of Software Engineering, so as to provide even better support regarding the *software* design, implementation, and maintenance aspects of large-scale traffic management DSS.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science and Innovation through projects CSD2007-0022 (Consolider-INGENIO 2010), TIN2006-14360-C03-02, and TIN2009-13839-C03-02, as well as the European Community through project EFRE Nr. 2-221-2007-0042.

References

Arasu, A., Babu, S., & Widom, J. (2003). Cql a language for continuous queries over streams and relations. In The 9th international conference on data base programming languages (DBPL).
Babcock, B., Babu, S., Datar, M., Motwani, R., & Widom, J. (2002). Models and issues in data stream systems. In L. Popa (Ed.), *PODS* (pp. 1–16). ACM.
Babu, S. & Widom, J. (2001). Continous queries over streams. In *SIGMOD record*.
Bandini, S., Bogni, D., & Manzoni, S. (2002). Alarm correlation in traffic monitoring and control systems: A knowledge-based approach. In F. van Harmelen (Ed.), *ECAI* (pp. 638–642). IOS Press.
Berstel, B. (2002). Extending the rete algorithm for event management. In *TIME*, pp. 49–51.
Bickel, P. J., Chen, C., Kwon, J., Rice, J., van Zwet, E., & Varaiya, P. (2007). Measuring traffic. *Statistical Science, 22*, 581.
Coral 8 (2008). <http://www.coral8.com/>.
Cuena, J., Hernández, J. Z., & Molina, M. (1996). Knowledge oriented design of an application for real time traffic management: The trys system. In W. Wahlster (Ed.), *ECAI* (pp. 308–312). Chichester: John Wiley and Sons.
Delic, K. A., Douillet, L., & Dayal, U. (2001). Towards an architecture for real-time decision support systems: Challenges and solutions. In M. E. Adiba, C. Collet, & B. C. Desai (Eds.), *IDEAS* (pp. 303–311). IEEE Computer Society.
Dresner, K. M., & Stone, P. (2005). Multiagent traffic management: an improved intersection control mechanism. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, & M. Wooldridge (Eds.), *AAMAS* (pp. 471–477). ACM.
Dunkel, J., & Bruns, R. (2008). Reference architecture for event-driven RFID applications. In Q. Z. Sheng, Z. Maamar, S. Zeadally, & M. Cameron (Eds.), *IWRT* (pp. 129–135). INSTICC PRESS.
Dunkel, J., Fernández, A., Ortiz, R., & Ossowski, S. (2009). Injecting semantics into event-driven architectures. In *11th international conference on enterprise information systems (ICEIS)*, pp. 70–75.
eDraft project (2009). edraft – event-driven architecture for traffic management. Technical report, Hannover University of Applied Sciences And Arts, Computer Science Department.
Ermert, L. (2009). Comparing jess and esper for event stream processing. Technical report, Hannover University of Applied Sciences And Arts, Computer Science Department.
ESPERTECH (2008). Esper reference documentation, version 2.0.0. Technical report, ESPERTECH.
French, S. (2000). *Decision analysis and decision support*. John Wiley & Sons.
GARTNET-GROUP (2007). Introducing the zero-latency enterprise. In *Research Note COM-04-37770*.
IBM, Websphere Business Events (2009). <http://www-01.ibm.com/software/integration/wbe/>.
ILOG JRules (2009). <http://www.ilog.com/products/jrules>.
Inoue, S., Shozaki, K., & Kakuda, Y. (2007). An automobile control method for alleviation of traffic congestions using inter-vehicle ad hoc communication in lattice-like roads. In *IEEE globecom workshops*.
JBoss Drools (2009). <http://www.jboss.org/drools>.
Jeffery, S. R., Alonso, G., Franklin, M. J., Hong, W., & Widom, J. (2006). A pipelined framework for online cleaning of sensor data streams. In L. Liu, A. Reuter, K.-Y. Whang, & J. Zhang (Eds.), *ICDE* (pp. 140–142). IEEE Computer Society.

Jess, the Rule Engine for the Java Platform (2009). <http://herzberg.ca.sandia.gov/>.
Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM, 21*(7), 558–565.
Luckham, D. (2002). *Power of events*. Addison-Wesley.
Oracle, Oracle Complex Event Processing (2009). <http://www.oracle.com/technology/software/products/cep/index.html>.
Ossowski, S., Hernández, J., Belmonte, M.-V., Fernández, A., García-Serrano, A., de-la Cruz, J.-L. P., et al. (2005). Ecision support for traffic management based on organisational and communicative multiagent abstractions. *Transportation Research Part C – Emerging Technologies, 13*, 272–298.
Ossowski, S., Hernández, J., Iglesias, C., & Fernández, A. (2002). Engineering agents systems for decision support. In P. Tolksdorf & Zambonelli (Eds.), *Engineering societies in an agent world III*. *LNAI* (pp. 234–274). Springer.
Paton, N. (1999). Active database systems. *ACM Computing Surveys, 31*(1), 63–103.
Roozemond, D. (1999). Using intelligent agents for urban traffic control systems. In *Proceedings of the international conference on artificial intelligence in transportation systems and science*, pp. 69–79.
Schiefer, J., Rozsnyai, S., Rauscher, C., & Saurer, G. (2007). Event-driven rules for sensing and responding to business situations. In *DEBS*. In H.-A. Jacobsen, G. Mühl, & M. A. Jaeger (Eds.). *ACM international conference proceeding series* (Vol. 233, pp. 198–205). ACM.
Sharon, G., & Etizon, O. (2008). Event-processing network model and implementation. *IBM Systems Journal, 47*(2), 321–334.
Streambase (2009). <http://www.streambase.com/>.
Tibco BusinessEvents (2009). <http://www.tibco.com/software/complex_event_processing/>.
Walzer, K., Breddin, T., & Groch, M. (2008). Relative temporal constraints in the rete algorithm for complex event detection. In *DEBS*. In R. Baldoni (Ed.). *ACM international conference proceeding series* (Vol. 332, pp. 147–155). ACM.
Wang, F., & Liu, P. (2005). Temporal management of rfid data. In K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P.-Å. Larson, & B. C. Ooi (Eds.), *VLDB* (pp. 1128–1139). ACM.
Wu, E., Diao, Y., & Rizvi, S. (2006). High-performance complex event processing over streams. In S. Chaudhuri, V. Hristidis, & N. Polyzotis (Eds.), *SIGMOD conference* (pp. 407–418). ACM.
Zimmer, D. & Unland, R. (1999). On the semantics of complex events in active database management systems. In *ICDE*, pp. 392–399.