# Extending Event-Driven Architecture for Proactive Systems

Fabiana Fournier
IBM Research – Haifa
Haifa University Campus
Haifa 3498825, Israel
+972 4 8296489
fabiana@il.ibm.com

Alexander Kofman
IBM Research – Haifa
Haifa University Campus
Haifa 3498825, Israel
+972 4 8281055
kofman@il.ibm.com

Inna Skarbovsky
IBM Research – Haifa
Haifa University Campus
Haifa 3498825, Israel
+972 4 8281330
inna@il.ibm.com

Anastasios Skarlatidis
Institute of Informatics and Telecommunications, NCSR "Demokritos"
Athens 15310, Greece
+30 210 6503217
anskarl@iit.demokritos.gr

## ABSTRACT

Proactive Event-Driven Computing is a new paradigm, in which a decision is not made due to explicit users' requests nor is it made as a response to past events. Rather, the decision is autonomously triggered by forecasting future states. Proactive event-driven computing requires a departure from current event-driven architectures to ones capable of handling uncertainty and future events, and real-time decision making. We present a proactive event-driven architecture for Scalable Proactive Event-Driven Decision-making (SPEEDD), which combines these capabilities. The proposed architecture is composed of three main components: complex event processing, real-time decision making, and visualization. This architecture is instantiated by a real use case from the traffic management domain. In the future, the results of actual implementations of the use case will help us revise and refine the proposed architecture.

## Categories and Subject Descriptors

C.0 [Computer Systems Organization]: General – *System architectures*; D.4.8 [**Operating Systems**]: Performance - *Modeling and prediction*; G.3 **[Mathematics of Computing**]: Probability and Statistics - *Distribution functions*, *Time series analysis*; H.1.2 [**Models and Principles**]: User/Machine Systems – Human factors; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving - *Uncertainty, fuzzy, and probabilistic reasoning*.

## General Terms

Performance, Design, Human Factors

## Keywords

Proactive computing, event-driven, real-time optimization, forecasting, uncertain and future events, visualization.

## 1. INTRODUCTION

Proactive Event-Driven Computing is a new paradigm ([6],[7], [9]), where a decision is neither made due to explicit users' requests nor as a response to past events, but is autonomously triggered by forecasting future states, either desired or undesired. The decisions and actions are often real-time in the sense that they are done under time constraints and require the exploitation of large amounts of historical and streaming data. The underlying motivation of proactive computing stems from social and economic factors, and is based on the fact that *prevention* is often more effective than *cure*.

Achieving this vision requires novel research in three different directions:

*Dealing with large quantities of data*. Massive volumes of historical data and massive streaming data have to be analyzed to forecast events. Most systems are not capable of handling big data in real-time because of scalability problems, the need to cleanse noisy data offline, or the difficulty in fusing different types of data coming from different sources online. The result is that most analyses are done on offline data, while online data is not leveraged for immediate operational decisions.

*Extending the state-of-the-art in event processing to deal with future events and uncertainty due to incomplete and noisy streaming data* [1]. The ability to process past events and forecast future ones makes proactive systems a compelling application area. But, the uncertain nature of future events requires a major leap in event processing systems.

*Devising methods for making near-optimal decision within time constraints*. The decision about which is the best action to take in proactive computing has two properties that differ from most contemporary decision support systems: (1) the decision should be taken on-line and under real-time constraints, which may dictate the use of approximation techniques and (2) The decision often entails autonomic actions, rather than providing only recommendations for human decision makers.

A proactive-driven architecture should satisfy the requirements above and provide an integrated platform that combines advanced event processing with dynamic forecasting capabilities leveraged towards online optimisation and decision-making. The proposed architecture presented in this paper, an outcome of the SPEEDD (Scalable ProactivE Event-Driven Decision making) project[1], exactly addresses this.

This paper is organized as follows: Section 2 briefly introduces the traffic management use case that will illustrate our proposed architecture. Section 3 presents a general overview of a proactive event-driven architecture, while Section 4 details the SPEEDD proactive event-driven architecture. We survey some related work in Section 5. We conclude the paper in Section 6.

## 2. ILLUSTRATIVE EXAMPLE

Proactive traffic management concerns the south ring of Grenoble, which is the main West to East artery around the city in France and a primary source for traffic congestion. The goal within this use case is to forecast traffic congestion before it



[1] http://speedd-project.eu

happens and, as a result, automatically act in order to attenuate it. This is done by forecasting traffic congestions a few minutes before they happen, and making decisions within a few seconds of the forecast about adjustment of traffic light settings and speed limits.

There are two sources of data in this use case: real data from sensors and synthetic data generated by a micro-simulator.

The input data (raw events) comes from 130 magnetic wireless Sensys sensors[2] buried in the road along the highway which can provide individual or aggregated data. Sensors are located in 19 collection points. Each collection point has a sensor per lane (slow and fast lane) and, where applicable, also has sensors on the on/off-ramps. Sensors provide data every 15 seconds. Such data can be either individual (concerning every single vehicle), or aggregated (over the 15-seconds time span). However, the individual and aggregated data cannot be collected simultaneously. Currently, aggregated data is being collected.

The simulator used for generating synthetic traffic data is the commercial micro-simulator by Aimsun[3]. The simulator has been calibrated using real traffic data from Grenoble South Ring.

# 3. PROACTIVE EVENT-DRIVEN ARCHITECTURE

Conceptually, we distinguish between the design time and runtime components.

At the *build or design time*, proactive applications are developed using *authoring tools* either directly by *experts* or with the help of *learning systems*. *Visualization tools* can be used to analyze the stored historical data during design time. By using the *authoring* and *visualization tools*, the experts may also annotate the historical data, in order to provide training examples for the machine learning algorithms. The products of the design time activities are event processing definitions and decision making configurations that will be deployed and executed at the runtime.

The *runtime* consists of four building blocks or components: event processing, forecasting, real-time decision making, and visualization tools. In general, raw events emitted by various event sources (e.g., traffic sensors) are processed by the complex event processing (CEP) engine and forecasted events serve for real-time decision making. The CEP engine processes raw as well as derived (detected and forecasted) events to detect and forecast higher-level events, or situations. These serve as triggers for the decision making component, which uses domain-specific algorithms to suggest the next best action to resolve or prevent an undesired situation.

Let's examine in more details the principles of each building block in the envisaged architecture:

The first building block required to facilitate proactive event driven computing is *a new kind of event processing component*. Event processing is an approach to software systems that is based on reaction to events, often under time constraints. It includes specific logic to filter, transform, or detect complex events and patterns in events as they occur [8]. The CEP component needs to be extended to cope with detecting and forecasting derived events under uncertainty.

The second building block facilitates *event recognition and forecasting*, that is, identifying events that either have occurred or are likely to occur in the near future. This is a key enabler of proactive computing, allowing decision-making to commence even before an event has been (completely) detected. This building block continuously refines event recognition and forecasting given the incoming, possibly noisy, data streams, in order to improve the recognition accuracy and probability estimations. Recognition and forecasting exploit models that can be created by human experts or through goal-driven supervised learning that exploits offline data available to the organization, or a combination thereof. A particularly challenging aspect of event forecasting is the temporal dimension. To facilitate precisely-informed online decision-making, forecasting should indicate not only which event will happen and with what probability, but also *when* it is expected to happen; more generally, forecasting should provide a probability distribution over the expected occurrence time.

The third building block enables the *event-based real-time decision making under uncertainty*. In order to realize proactivity and support autonomous or semi-autonomous decision-making, a body of tools is required that can exploit the forecast models and state predictions as a basis for decision-making. These tools will have to properly consider the nature and degree of uncertainty in the models' forecasts when generating decisions.

The forth building block, the *visualization component* (or *dashboard*) supports the human interpretation of decisions made in runtime. It facilitates decision making process for business users by providing easily comprehensible visualization of detected or forecasted situations along with output of the automatic decision making component – a list of suggested actions to deal with the situation. The proposed architecture can be run in open, closed, or hybrid loop mode. In case of the open loop, the user can approve, reject, or modify the action proposed by the automatic decision maker. The closed loop operation does not require user's approval, the action is performed automatically. A hybrid mode where some types of actions are taken automatically while other types require human attention is also supported.

With the quantity of events, the volume of historical data, and the complexity of applications all growing fast, it is vital that the proposed architecture also exhibit scalable behavior. Scalability has several dimensions, including scalability in streaming events, scalability in volume of historical data, scalability in amount of data sources and sinks, scalability in amount of processing elements, and scalability in terms of physical infrastructure.

# 4. SPEEDD ARCHITECTURE

In the scope of the SPEEDD project a proactive event-driven architecture has been proposed [10] that follows the conceptual architecture presented in Section 3 and consists of all the building blocks introduced. In the following sections we describe this architecture using the traffic management scenario.

## 4.1 System Requirements

The requirements for the current prototype are derived from the traffic management use case. The detailed requirements can be found in [2].

The prototype should provide authoring tools that could be applied to the historic data in order to derive event pattern definitions and decision models to be deployed in runtime, as well as a scalable runtime system capable of detecting and predicting

[2] http://www.sensysnetworks.com

[3] http://www.aimsun.com/wp

important situations (traffic conditions) and issuing automatic actions aimed at preventing undesired situations (congestions).

For the traffic management scenario, the projected throughput is 2000 sensor readings per second (computed based on the amount of sensors and the report frequency, assuming aggregated readings sent every 15 seconds by each of the 130 Sensys sensors installed along the Grenoble South Ring).

In terms of integration with external systems the following is required:

- Replay historic events from text files or a database.
- Receive sensor reading messages generated by the micro-simulator.
- Provide a mechanism to log output events and actions to a log for subsequent research.
- Provide a mechanism to connect to the traffic micro-simulator for updating the simulator configuration – action simulation.

## 4.2 SPEEDD Runtime Architecture

The architecture of the runtime part of SPEEDD follows the Event-Driven Architecture paradigm [12]. This approach facilitates building loosely coupled highly composable systems, as well as provides close alignment with the real world problems, including our representative use case. Every component functions as an event consumer, or an event producer, or a combination of both. The event bus plays a central role in facilitating inter-component communication which is done via events. Figure 1 shows the event-driven architecture for SPEEDD where the runtime part is represented as a group of loosely-coupled components interacting through events. The event bus serves as the communication and integration platform for SPEEDD runtime.

Input from the operational systems (traffic sensor readings) are represented as events and injected into the system by posting a new event message to the event bus. These events are consumed by the CEP runtime. The derived events representing detected or forecasted situations that CEP component outputs are posted to the event bus as well. The decision making module listens to these events so that the decision making procedure is triggered upon a new event representing a situation that requires a decision. The output of the decision making represents the action to be taken to mitigate or resolve the situation. These actions are posted as action events. The visualization component consumes events coming from two sources: the situations (detected as well as forecasted) and the corresponding actions suggested by the automatic decision components. Architecturally, there is no difference between these two – both are events that the dashboard is 'subscribed to', although having different semantics and presented and handled differently. The user can accept the suggested action as is, modify the suggested action's parameters, or reject it (and even decide upon a different action). In the case where an action is to be performed, the resulting action will be sent as a new event to the event bus so that the corresponding actuators are notified.

Specifically, Figure 2 shows the SPEEDD runtime architecture for the traffic management use case, including the technology platforms used to implement the architecture. In the following subsections we describe the details of the runtime architecture including the design of each component and its technology implementation.

### *4.2.1 Event Bus*

The technology chosen for the event bus component is Apache Kafka [16]. It provides a scalable, performant, and robust messaging platform that matches SPEEDD requirements. To implement routing of the events to event consumers we build upon the topic-based routing mechanism provided by Kafka.

To allow scalable processing of massive stream of messages at high throughput, Kafka provides the partitioning mechanism. Every topic can be partitioned into multiple streams that can be processed in parallel, while every partition can be managed in a separate machine. There may be more than one replica for every partition, thus providing resilience in case of failures.

In SPEEDD we exploit Kafka partitioning to build a scalable and fault-tolerant event bus. The topic that receives the biggest incoming traffic is *speedd-in-events* where all the input events are sent. The decision about the partitioning mechanism to use is use-case specific as we want to achieve nearly uniform distribution of load over different partitions. Below, we describe the partitioning approach for our use case, providing the rationale for the design decisions. It is important to mention, though, that we may change the final partitioning mechanism based on the performance experiments on real and simulated data. We will be able to do that at any stage of the project development, thanks to the highly extensible and customizable partitioning framework that Kafka provides.

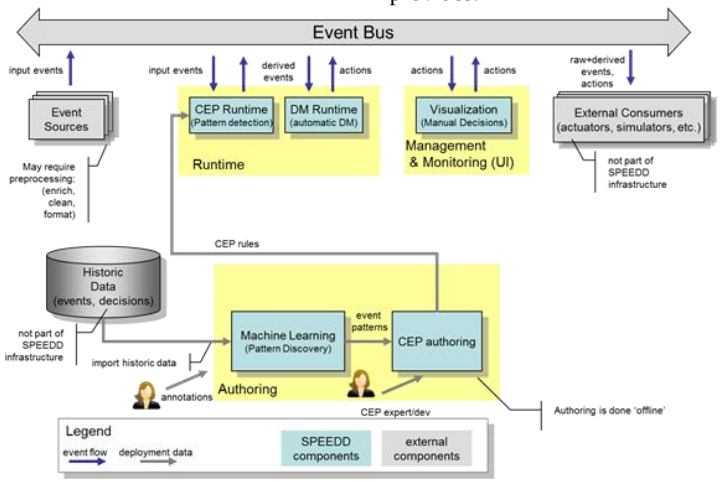


**Figure 1. SPEEDD Event-Driven Architecture**

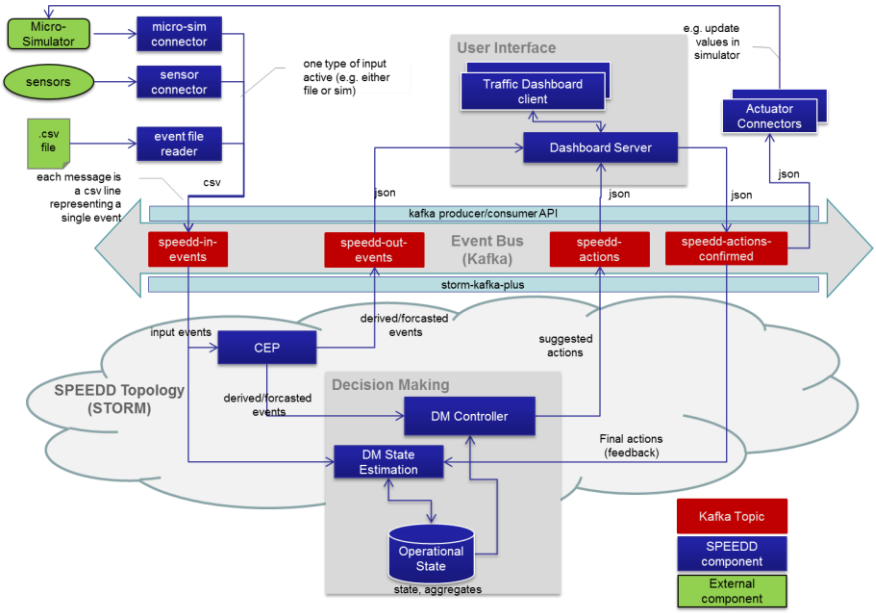


**Figure 2. SPEEDD Runtime Event-Driven Architecture (Traffic Use Case)**

#### *4.2.1.1 Partitioning for the Traffic Use Case*

Assuming that we get relatively equal amount of events produced by every sensor, we could partition sensor reading events based on the sensor id. This should result in uniform distribution of the messages to partitions, which provides horizontal scalability of the topic.

#### *4.2.1.2 Ordering of events*

Kafka guarantees that the order of events submitted to a topic's partition is preserved within same partition – the consumers will receive them in the same order. However, the order is not guaranteed across partitions. In our case, this should not be an issue because the CEP component takes care of the out-of-order events as long as the delay between the event and its preceding event that arrives after that event is not too long – this assumption should be valid with Kafka.

#### *4.2.1.3 Storm-Kafka Integration*

SPEEDD event processing and decision making components run on top of Apache Storm [25], a distributed scalable stream processing infrastructure.

Integration between Storm streaming platform and our Kafka-based event bus is done based on the Storm-Kafka-Plus project[4]. Storm-Kafka-Plus provides two building blocks. KafkaSpout listens on a Kafka topic and creates a stream of the tuples. KafkaBolt posts incoming tuples to a configured topic. There is an extensible mechanism for serialization and deserialization of tuples to messages and vice versa.

### *4.2.2 Event/Data Providers*

Event providers provide the input interface of SPEEDD runtime with the external world. Every event that occurs in the external world that should be taken into account by SPEEDD to detect or predict an important business situation should be sent to the *speedd-in-events* topic on the event bus as a message representing the event.

As it is illustrated in Figure 2, events for the traffic use case may come from traffic sensors (magnetic wireless Sensys sensors buried in the road), micro-simulator (synthetically generated data), as well as historic data (collected data from sensors).

To enable processing of events generated by any of the above sources, a connector should be developed. The connector uses source-specific integration mechanism to read the data from the event sources and send them to SPEEDD event bus using Kafka producer API.

We define three connector types corresponding to the types of the event sources, that is, file-reader (replay past events from a file) sensor, and micro-simulator connectors.

### *4.2.3 Action Consumption – Actuators/Connectors*

The outcomes of SPEEDD are actions that should be applied in the operational environment to resolve a problem or prevent a potential problem. According to the event-driven architecture principles, actions are represented as outbound events and are available to every interested party to receive and process them. The actuators connectors are interface points in SPEEDD architecture responsible for listening to the speedd-actions-confirmed topic for new actions and connect to operational systems to execute respective operations.

As it is not planned to connect SPEEDD prototype to the traffic operational systems running in production mode, the detect→decide→act loop will be implemented and tested using the AIMSUN micro-simulator [2]. The traffic actuator connector will listen to the outbound action events (speedd-actions-confirmed topic on the event bus) and execute operations supported by the micro-simulator, e.g., update speed limits, set ramp metering rates, etc. The integration with the event bus for actuators is based on the Kafka consumer API.

[4] https://github.com/wurstmeister/storm-kafka-0.8-plus

### 4.2.4 Complex event processing component

The main role of the CEP component is to detect events and derive situations to feed the decision module, so proactive actions can be taken. To this end, the CEP component needs to deal with uncertainty in the input, as well as the output events.

We use the IBM Proactive Technology Online (Proton) research asset as the CEP engine in SPEEDD. This engine has been released as open source as an outcome of the FI-WARE project[5] and it is extended to cope with predictive capabilities in the scope of the SPEEDD project.

Proton receives raw events, and by applying patterns defined within a context on those events (we follow the terminology in [8]), computes and emits situations (derived events emitted to consumers). Proton is platform-independent, as it is implemented in Java. The architecture is modular and consists of the following components:

*Adapters* – communication of Proton with external systems

*Parallelizing agent-context queues* – for parallelization of processing of single event instance, participating in multiple patterns/contexts, and parallelization of processing among multiple event instances.

*Context service* – for managing of context's lifecycle – initiation of new context partitions, termination of partitions based on events/timers, segmenting incoming events into context groups which should be processed together.

*EPA manager* – for managing Event Processing Agent (EPA) instances per context partition, managing its state, pattern matching, and event derivation based on that state.

SPEEDD will take advantage of the adaptation of the standalone architecture of Proton to a distributed architecture done in the scope of the FERARI FP7 EU project[6], and will apply the Proton on Storm version of the engine. It is important to note that, while Storm offers an open programming model so developers can add the logic to address complex event driven applications, the resulting implementation is custom to a single application and not a generic re-usable solution. Furthermore, the inclusion of uncertainty requires additional specific coding to deal with. In the architecture proposed, we make use of a generic event processing system that provides the necessary building blocks to build generic event driven applications with the presence of uncertainty.

The Proton architecture on top of Storm preserves the same logical components as are present in the standalone architecture: the queues, the context service and the EPA manager, which constitutes the heart of the event processing system. However the orchestration of the flow between the components is a bit different, and utilizes existing Storm primitives for streaming the events to/from external systems, and for segmenting the event stream.

### 4.2.5 Decision making component

As aforementioned, the aim of the real-time decision making building block is to provide a body of proactive event-driven decision-making tools, which exploit the detected or forecasted events of the CEP. The Decision Making (DM) module receives as inputs the detected, derived, and forecasted events and emits control actions or appropriate suggestions. Therefore, it functions both as an event consumer and as an event producer at the same time.

In this sense, decision making is the task of finding the optimal response to a specific situation, which is described by the detected or forecasted events. It is naturally represented as a parametric optimization problem. The main task of decision making is to solve this optimization problem, which can be accomplished in two conceptually different ways:

The parametric optimization problem is solved offline such that an explicit solution is obtained. Note that this is a "difficult" task, since an optimal answer to any situation that might arise during operation needs to be computed. If such an explicit solution can be obtained, it takes the form of a feedback rule, e.g. a linear controller K(s) or state feedback - K*x. Therefore, it can be efficiently implemented in a unified architecture using the existing SPEEDD components (e.g., as a Storm Bolt).

The construction of an explicit solution may be computationally intractable for certain problems. In such a case, the solution to multiple distinct instances of the optimization problem needs to be computed at runtime. In contrast to the first case, in which only the evaluation of a feedback rule is required, the algorithmic solution of an optimization problem is not trivial and it is not tractable to solve such a problem within the stream processing environment adopted in SPEEDD (Storm). We, therefore, assume the existence of a use-case specific "optimization black-box" outside the actual SPEEDD framework, which can be queried whenever such a decision is required.

In our illustrative example of freeway ramp-metering (regulating the traffic inflow on a freeway in order to maximize throughput), a low-level ramp metering controller receives measurements of the local traffic density and the local traffic flows, as well as notifications about detected or predicted congestion queues. It then emits a recommendation to change the ramp metering rates accordingly. For a network of interaction freeways, a network-wide planning algorithm can be used for coordination purposes, implemented as an external oracle that can be queried.

Since a road network is naturally a spatially distributed system, the architecture of the decision-making module reflects this structure. Specifically, the module is directly and efficiently implemented as Storm bolts in a distributed manner. Preliminary theoretical results suggest that such local controllers may perform asymptotically optimal with regard to flow maximization for a single freeway; however, coordination is required to achieve optimal operation of more complex road networks. Network-wide planning can be superimposed by querying an external black-box.

### 4.2.6 Dashboard component

As aforementioned, the proposed event-driven architecture can be run in an open, closed, or hybrid loop mode. In the traffic management use case we only deal with open or hybrid modes, i.e., we don't have fully automatic actuators for the decisions. The closed mode implies connecting the SPEEDD prototype to the actual production systems and, therefore, out of the scope of the project.

In our current scenario, operators interact with the outputs of the SPEEDD modules through a User Interface (UI). The Dashboard Client communicates, via the Dashboard Server, with the composite systems in the SPEEDD architecture. Operators can accept, respond to, or make suggestions and control actions. Actions taken by operators via the UI are fed back into the SPEEDD runtime as events, thus allowing for the seamless

[5] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Architecture

[6] http://www.ferari-project.eu

integration of expert knowledge and the outputs of complex algorithms.

The Dashboard Server component is based on Node.js [24] asynchronous programming framework. The server functions as a Kafka consumer and producer. The consumer listens for broadcasted messages in the Event Bus under the following topics: *speedd-out-events* and *speedd-actions*. The producer broadcasts messages under the topic *speedd-actions-confirmed* (see section 4.2.1)

The Dashboard Client is designed to provide the user with a clear picture of the current state of the world. It achieves the picture of the current state by aggregating sensor readings in human readable form, current states of the control equipment available (e.g., speed limit signs, message signs, lanes), current events identified by the CEP module, and displays of the automated control events produced by the DM unit (e.g., ramp metering rates). Furthermore, it aims to support the decision-maker by highlighting events which might require attention along with corresponding suggested mitigating strategies.

## 4.3 SPEEDD Design Time Architecture

In general, there exist two methods to define the rule patterns for a CEP application: machine learning and experts. In the first, the patterns are learnt automatically by a computer program, while in the second, they are given by an external entity; usually a subject expert matter specialized in the domain. It is also possible to combine between these two methods.

Historic data used at design time contains raw events reported during the observed period along with annotations provided by domain experts. These annotations mark important situations that have been observed in the past and should be detected automatically in the future. Domain experts can apply tools and methodologies provided by SPEEDD authoring toolkit to extract derived event definitions from the annotated event history. This is a semi-automatic process involving applying machine learning tools to extract initial set of patterns, then further enhanced and translated with help of the domain experts into deployable CEP artefacts.

Due to the dynamic nature of the proactive traffic management application, the knowledge base of event pattern definitions may require to be refined or enhanced with new ones. Manual creation of event definitions is often a tedious and cumbersome process, thus we employ machine learning techniques to semi-automatically create event pattern definitions by analyzing historical data.

We employ the Probabilistic Event Calculus [23] that combines temporal logic-based formalization with probabilistic modelling. The logic-based representation allows to compactly define relations between events and incorporate existing domain knowledge, while probabilistic modelling allows to naturally handle uncertainty. For the implementation of the machine learning algorithms, we extend the open-source framework LoMRF[7] with state-of-the-art scalable probabilistic inference and incremental learning methods [14]. LoMRF is developed in Scala[8], which compiles to Java bytecode and thus works seamlessly with any other Java-based framework.

Additionally, for scalability LoMRF employs the high-performance parallel processing framework of Akka Actors[9].

The resulting output of the machine learning algorithms is composed of a set of text-formatted files that contain the event pattern definitions. Thereafter, the resulting rules are parsed by the "rtec2proton" translator and converted semi-automatically to JSON formatted Proton EPN definitions. All EPN definitions are then reviewed and manually refined by domain experts using Proton's authoring tool. The output of this process is a JSON file containing the EPN definition.

# 5. RELATED WORK

Proactive applications have been developed in an ad-hoc manner for several years; some examples include proactive security systems [5], proactive routing in mobile ad-hoc wireless [17], proactive network management with failure handling [11], proactive service level agreement negotiation in service oriented systems [18], proactive caching [15], and proactive management in logistic processes [19] and [9]. However, the lack of a generic paradigm to develop proactive event-driven applications makes it difficult for this capability to spread.

One of the main ingredients for proactive event driven computing is the ability to deal with uncertainty in the events. Despite uncertainty handling has been recognized as one of the most critical and relevant aspects in the area of CEP, it still remains an open issue [1]. Only a few solutions have been proposed, and most of them are tailored to a specific application domain [4]. Examples of previous works can be found in [4], [20], [22], [26], [27] and [28]. Existing CEP approaches examine three major types of uncertainty that may be present in the events that are fed in a CEP system: uncertainty in event content, in the event occurrence, and in the rules. Our CEP component must support these three types. Furthermore, learning event rules in the presence of uncertainty is also an open research area [1].

In terms of real-time optimization techniques, the state-of-the-art is that optimization techniques are being activated mostly off-line and use a variety of optimization methods that fit different assumptions: robust (worst-case) optimization, stochastic optimization, and optimization methods based on black-box models (e.g., [3], [13] and [21]). Our main challenge is to develop real-time proactive planning tools for proactive applications using these optimization methods within an event-based planning framework.

# 6. SUMMARY AND FUTURE WORK

Event-driven architecture is a software architecture pattern promoting the production, detection, consumption of, and reaction to events. We describe how we extended this architecture from being reactive to proactive, by incorporating capabilities for forecasting and real-time decision making.

The proposed architecture is instantiated by a real use case from the traffic management domain. Although driven by the use case requirements in the SPEEDD project, the proposed architecture is generic and can be applied to any domain that requires proactive event-driven computing.

We are currently working on a first implementation of the use case based on the proposed architecture. Future work includes integration of offline historic data and online streaming data as

[7] https://github.com/anskarl/LoMRF

[8] http://www.scala-lang.org

[9] http://akka.io

well as refinements to the proposed architecture as result of the implementation.

## 7. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme FP7/2007-2013 under grant agreement 619435 (SPEEDD).

## 8. REFERENCES

[1] Artikis A., Etzion O., Feldman Z., and Fournier F. 2012. Event Processing under Uncertainty. In *Proceedings of the sixth ACM conference on Distributed Event-Based Systems (DEBS'12).*

[2] Baber C., Bellicot I., Canudas de Wit C., Cooke N., Garin F., Grandinetti P., Hempel A., Kibangou A., Morbidi F., and Schmitt. M. User requirements and scenario definition. Accessible at http://www.speedd-project.eu/sites/default/files/D8.1-UserRequirements_final.pdf

[3] Ben-Tal, A., Boyd, S., and Nemirovski. A. 2006. Extending Scope of Robust Optimization: Comprehensive Robust Counterparts of Uncertain Problems. *Mathematical Programming*, 107:1-2, 63-89.

[4] Cugola G., Margara A., Matteucci M., and Tamburrelli G. 2014. Introducing uncertainty in complex event processing: model, implementation, and validation. *Computing* 1–42.

[5] Dolev S., Kopeetsky M., and Shamir A. 2011. RFID Authentication Efficient Proactive Information Security within Computational Security. *Theory of Computing Systems*, 1-18.

[6] Engel Y., Etzion O., and Feldman Z. 2012. A Basic Model for Proactive Event-Driven Computing. In *Proceedings of the sixth ACM conference on Distributed Event Based Systems (DEBS'12)*.

[7] Engel Y. and Etzion O. 2011. Towards proactive event-driven computing. In *Proceedings of the fifth ACM conference on Distributed Event Based systems (DEBS'11)*.

[8] Etzion O. and Niblett P. 2010. *Event Processing in Action.* Manning Publication.

[9] Feldman Z., Fournier F., Franklin R., and Metzger A. 2013. Proactive event processing in action: A case study on the proactive management of transport processes, in *Proceedings of the Seventh ACM International Conference on Distributed Event-Based Systems (DEBS'13).*

[10] Fournier F., Kofman A., Morar N., Schmitt M., Skarbovsky I., and Skarlatidis A. 2014. The Architecture Design of the SPEEDD Prototype. Accessible at http://www.speedd-project.eu/sites/default/files/D6.1-Architecture_Design_of_SPEEDD_Prototype-v1.0a.pdf

[11] Fu S. and Xu C.Z. 2007. Exploring event correlation for failure prediction in coalitions of clusters. *In Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC'07)*, 1-12.

[12] Hohpe G. Programming without a call stack – Event-driven Architecture. 2006, [Online], at: http://www.eaipatterns.com/docs/EDA.pdf

[13] Hokayem P., Chinquemani E., Chaterjee D., Ramponi F., and Lygeros J. 2012. Stochastic receding horizon control with output feedback and bounded controls. *Journal Automatica*, 48, 1, 77-88.

[14] Huynh, T. N., & Mooney, R. J. 2011. Online structure learning for markov logic networks. *In Machine Learning and Knowledge Discovery in Databases*, 81-96.

[15] Kohler M. and Fies R. 2009. ProActive Caching-A Framework for Performance Optimized Access Control Evaluations. In *Proceedings of IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY 2009).*

[16] Kreps, J., Narkhede N., and Rao J. 2011. Kafka: A distributed messaging system for log processing. In *Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB).*

[17] Kunz T. and Alhalimi R. 2010. Energy-efficient proactive routing in MANET: Energy metrics accuracy. *Ad Hoc Networks*, 8(7), 755-766.

[18] Mahbub K. and Spanoudakis G. 2010. Proactive SLA Negotiation for Service Based Systems. In *Proceedings of the 6th World Congress on Services (SERVICES-1).*

[19] Metzger A., Franklin R., and Engel Y. 2012. Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case. In *Proceedings of the Annual SRII Global Conference (SRII)*, 313-322.

[20] Ré C., Letchner J., Balazinksa M., and Suciu D. 2008. Event queries on correlated probabilistic streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*, 715–728.

[21] Shapiro, A. 2008. Stochastic programming approach to optimization under uncertainty. *Mathematical Programming*, 112.

[22] Shen Z., Kawashima H., and Kitagama H. 2008. Probabilistic event stream processing with lineage. In *Proceedings of Data Engineering Workshop (DEWS' 08).*

[23] Skarlatidis A., Paliouras G., Artikis A. and Vouros G. 2014. Probabilistic Event Calculus for Event Recognition. *ACM Transactions on Computational Logic (TOCL)*, to appear.

[24] Tilkov S. and Vinoski S. 2010. Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing 14(6)*, 80-83.

[25] Toshniwal, A., Taneja S., Shukla A., Ramasamy K., Patel J. M., Kulkarni S., Jackson J., et al. 2014. Storm@twitter. In *Proceeding of the 2014 ACM SIGMOD international conference on Management of data (SIGMOD '14)*, 147-156

[26] Wasserkrug S, Gal A., Etzion O., and Turchin Y. 2012. Efficient processing of uncertain events in rule-based systems. *IEEE Transactions on Knowledge and Data Engineering*, 24(1), 45–58.

[27] Wasserkrug S., Gal A., Etzion O., and Turchin Y. 2008. Complex event processing over uncertain data. In *Proceedings of the Second ACM conference on Distributed Event-Based Systems* (DEBS '08), 253–264.

[28] Zhang H., Diao Y., and Immerman N. 2010. Recognizing patterns in streams with imprecise timestamps. *Proc. VLDB Endowment*, 3(1-2), 244–255.