# A SOA Approach to Delay and Jitter Tolerant Distributed Real-Time Complex Event Processing

Per Lindgren, Rumen Kyusakov, Jens Eliasson, Henrik Mäkitaavola, Paweł Pietrzak
Department of Computer Science, Electrical and Space Engineering
Luleå University of Technology,
971 87 Luleå, Sweden
Email: {per.lindgren, rumen.kyusakov, jens.eliasson, henrik.makitaavola, pawel.pietrzak}@ltu.se

***Abstract*—The combination of Service Oriented Architectures (SOAs) and Complex Event Processing (CEP) is gaining momentum for event centric management and processing of information in complex distributed systems (e.g., business automation). Whereas systems for factory automation have traditionally been deployed using dedicated buses and proprietary (often scan based) protocols, a recent trend in process automation is towards adopting open internet based technologies and event based communication. This trend is driven by the increasing number and capabilities of devices used for monitoring and control, and the increased flexibility, maintainability and price/performance gains expected from IP (potentially SOA/CEP) enabled systems.**

**In this paper we discuss the challenges involved to apply SOA and CEP to the field of factory automation. In particular, real-time aspects are highlighted, both w.r.t. to accurate time-stamping of physical events in a distributed system, as well as end-to-end timing including communication and CEP processing.**

**We approach the challenges by an architecture combining state-of-the-art synchronisation mechanisms for wired and wireless networks together with real-time communication and distributed query processing based on the notion of time constrained reactions. We discuss the impact of synchronisation inaccuracies and delays introduced by processing and communication, and present a method for implementation of safe potential- and certain matches.**

## I. Introduction

Systems, and systems of systems are getting increasingly complex, which calls for methods to facilitate design, reusability, maintenance and modifiability. In the area business automation, Service Oriented Architectures (SOAs) and Complex Event Processing (CEP) are gaining momentum, offering event centric management and processing of information in complex distributed systems. Underlying technologies such as web services and standardised protocols have the potential of providing cost efficient, future proof alternatives to proprietary solutions and closed source.

This trend is about to reach into the area of process monitoring and control, which involves an ever increasing amount of sensor and actuator devices, see e.g. [1], [2], [3] and the work of Kyusakov et al. [4]. Whereas such systems traditionally has been deployed using dedicated buses and proprietary (often scan based) protocols, advances in technology allow for such devices to operate autonomously and, through communication, cooperatively carry out tasks in a distributed and (potentially) event driven manner.

This opens up for supporting local intelligence into autonomous subsystems as well as reducing the amount of raw (unprocessed) data needed to be communicated over potentially bandwidth limited channels (e.g., wireless links). However, SOA and CEP technologies have been developed without direct consideration on timing requirements for the processing of queries, hence available solutions are not directly applied to systems for real-time monitoring and control. Our goal is to identify the challenges involved to treat end-to-end requirements for distributed CEP queries, to propose and architecture and outline technical details targeting the challenges. In contrast to other approaches we strive for standards based middleware and communication protocols for (complex) event delivery.

Firstly, in order to deal with complex event queries in the context of a distributed system, a common notion of time needs to be established (at least within subsystems for which events contribute to common queries). In Section II we discuss state-of-the-art technologies for wired and wireless synchronisation schemes. Secondly, the events in a distributed system needs to be communicated. In Section II-A we discuss state-of-the-art technologies for light weight SOAs and wired and wireless real-time communication protocols and in Section II-C we review a recent approach to real-time complex event processing, which based on the notion of time contained reactions allows time properties to be either explicitly specified or derived in a compositional manner.

In Section III we propose an architecture combining mentioned state-of-the-art techniques to accomplish a distributed real-time CEP system. The impacts of processing and communication delays as well as synchronisation errors are discussed in detail. We introduce the concept of potential and certain matches. Given that time-stamp errors (jitter) within a (sub)system are bound a potential match occurs when we cannot rule out the possibility of a match. A certain match occurs when we can rule out the risk of a false positive. Finally we show how the query processing can be implemented to provide end-to-end real-time guarantees under processing and communication delays in a distributed setting.

In Section IV we discuss related work and the in Section V we summarise contributions and discuss topics for future research.

## II. Background

### A. Service-oriented Distributed Computing

SOA denotes the use of loosely coupled distributed functions (services) that support dynamic discovery, eventing and composition. There are different approaches for implementing SOA - both proprietary and based on open standards. Today, industry and research communities alike are focusing on the web service technologies. The main drive behind this is the open access to the specifications and to open source tools and development kits for building web service applications in different programming languages. Although the core technologies of the web services are the same (the standard specifications used for the web - TCP/IP, HTTP, DNS, XML etc.) there are two distinct types of web services. The first one is based on the Simple Object Access Protocol (SOAP) and is widely adopted in enterprise applications. The second type is the RESTful WS [5] that is based on the notion of distributed resources and CRUD operations (Create, Read, Update and Delete).

The access to diverse information and the possibility to integrate functionality from different domains, e.g. enterprise and process automation, make the application of web services beneficial for many applications that were traditionally based on proprietary solutions. The specific requirements of the industrial process monitoring and control, including strict timing and the use of resource constrained devices, limit the possibility of using standard web technologies such as HTTP and XML that are heavy to process and require high bandwidth network links.

However, emerging specifications for efficient RESTful web services that use binary coding are soon to change that [6]. CoAP is an IETF application protocol specification that easily integrates with HTTP and provides asynchronous message exchange and build-in light-weight eventing and subscription mechanism. Using efficient structured representation of the service payload is enabled by Efficient XML Interchange (EXI) format [7] developed by W3C. EXI uses concepts from information theory and formal languages and the knowledge of the structure of the XML documents to achieve compact representations.

Using light-weight web services based on CoAP/EXI enables new opportunities for offloading the computations from the devices to cloud architectures. An example for this approach is the *thin server architecture* presented by Kovatsch et al. [8].

### B. Time Synchronisation

Distributing accurate time can be achieved in a number of different ways. The GPS system can for example be used to obtain a global time with a theoretical resolution down to the order of nanoseconds [9]. Accuracy is in practise limited to the interface and software implementation in between the host and the GPS receiver. The use of GPS provides a very high accuracy clock synchronisation and a relatively low cost, and works well on mobile devices as well. However, GPS does require line of sight with satellites and does now perform well, or even at all, indoors. Another mean of synchronising clocks is the use of radio-based solutions.

The Network Time Protocol (NTP) [10], is an IP-based protocol for time synchronisation between computers and other devices. NTP can synchronise clocks with about one millisecond accuracy [11]. Recent developments in version 4, NTPv4 [12], claim that resolutions down the tens of microseconds are possible to achieve.

Precision Time Protocol (PTP) [13] is another distributed time synchronisation protocol mainly aimed at industrial applications, e.g. process monitoring and control. PTP fills the gap between hardware-based solutions such as GPS, and general-purpose communication protocols such as NTP. PTP is designed to meet requirements to synchronise a local network down the order of microseconds.

In a typical installation, a GPS-receiver can be used to synchronise a server's master clock with global time. PTP or NTPv4 can then be used to synchronise other servers, computes and devices in the local network. This combination of hardware- and software-based solutions provides a cost-efficient, yet very accurate, solution. In many cases, it is sufficient to have a very accurate local distributed time, in this case both NTPv4 and PTP have adequate properties for most applications.

### C. Complex Event Processing

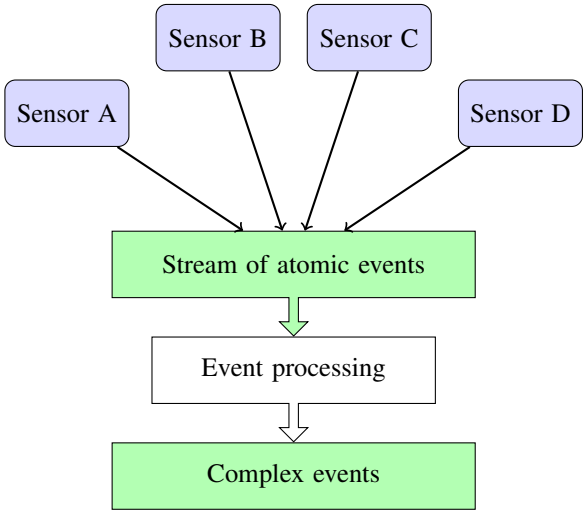



Fig. 1. Basic event processing

An event is a record of something that has (or has not) occurred. Basic events, also referred to as *atomic events* hold unprocessed (raw) data. In the context of process automation systems one can think of atomic events as of sensor readings, for instance as depicted in Figure 1.

Atomic events form an *event stream* which is a subject to *event processing*. Event processing, matches event pattern(s) in the stream, filters interesting events and possibly transform the detected pattern into a new (complex) event. Events can be pattern-matched with respect to their temporal properties or

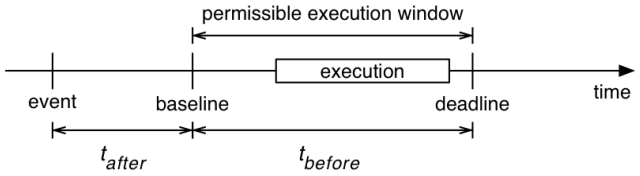



Fig. 2. Permissible execution window for a message.

values that they carry in their payloads. Such CEP queries are often static (operating on streaming data to detect some pattern or scenario), in contrast to traditional SQL like queries on data bases, which typically are dynamic (operating on relatively static data). This is manifested in event query languages, such as CEDR [14], [15] and ETALIS Language for Events (ELE) [16] and the combined event and stream processing language EP-SPARQL [17].

### D. Concurrent Reactive Objects

Event driven (reactive) systems with (optional) timing constraints can be modelled and implemented using the notions of Concurrent Reactive Objects (CROs) and time constrained reactions. Resources (such as memory, I/O, and other stateful elements) are managed in terms of objects, which act as monitors serialising access to the object's methods. The reaction to an event is defined in terms of a message identifying the receiver object $O$, method $m$ and optional timing constraints (baseline $B$ and deadline $D$ for the message execution). For external events, $B$ is the arrival time, while internal events may be postponed relative to the sender using a baseline offset, see Figure 2. Methods execute sequentially, run-to-end and may in turn create new messages, $\mathsf{ASYNC}(O, m, B, D))$ (non blocking), and $\mathsf{SYNC}(O, m)$ (blocking). Whenever timing windows overlap, reactions (messages) can be carried out concurrently. Trough built in concurrency and serialisation the CRO model provides flexibility for (potentially parallel) scheduling while allowing sequential behaviour to be enforced where needed (e.g., protecting shared resources, protocol interaction etc.)

The CRO model is manifested e.g., in the Timber programming language [18], [19], [20]. the TinyTimber [21] and REKO tools [22], which allow CRO models to be implemented directly in C.

### E. Real-Time Event Processing using CRO

The notion of time appearing in languages such as CEDR is used for the casual conditions for the query. However, in the setting of a measurement and/or control system, there is typically a need to define the timing properties for the processing of the query (in order output data and/or perform control).

In this section, we briefly review a real-time event processing mechanism, building on the translation of CEDR quires into CRO objects. We refer the reader to [23] for an in depth description and [24] for the extensions to overlapping events, real-time constraints and non-deterministic scheduling.

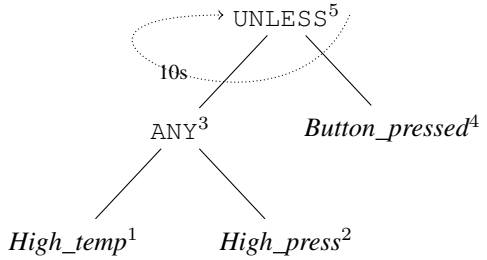
Consider the example query:

**EVENT** $Master\_alarm$
**WHEN** UNLESS(ANY($High\_temp, High\_press$) **AS** $x$, $Button\_pressed$, 10 seconds)
**WHERE** $x.Tank\_Id = 'Tank\#1'$ **OR** $x.Tank\_Id = 'Tank\#3'$
**OUTPUT** $Alarm\ x.Tank\_Id$

Its event recognising **WHEN** clause is shown below. In order to avoid ambiguity every subexpression $p$ in a **WHEN** clause is labeled with a label $\ell$, written as $p^{\ell}$.

**WHEN** UNLESS(ANY($H\_temp^1, H\_press^2$)$^3$ **AS** $x$, $Button\_pressed^4$, 10 seconds)$^5$

An event pattern like the one above can be decomposed into subexpressions, each of them being an event pattern itself. The decomposition is illustrated below.




Every node corresponds to an event pattern, with possible further subexpressions. Atomic events are leaves in the pattern expression.

The approach to process CEP queries under timing requirements associates each event (either primitive or complex) with a message, and each event pattern (node) with a CRO (object). The object state holds the state of processing, while the methods `start, accept, drop` typically are used to activate recognition, accept an event, and inactivate recognition respectively.

In the following we will use the Timber language to define the CRO model for the CEP. As an example we study the definition of `unless` as depicted in Figure 3.

For the constructor, `e` is the recipient object (parent in the query tree), `w` is the window size, whereas `d1` and `d2` defines the timing requirements on the query (as discussed later). On creation (not depicted) `new unless ...` returns an interface with the methods (actions) {`start, accept, drop`}. In the working example `start` is triggered by the `ANY` event, while `drop` is triggered from the abort alarm button *Button_pressed*. (`accept` is triggered internally as discussed below.)

In brief, `start` message triggers a postponed `accept` message, which in turn determines whether a `e.drop` (no alarm) or `e.accept` (alarm) message should be emitted dependent on the occurrence of abort message(s) within the window `w`. We accomplish this through time constrained reactions using the primitives provided by the Timber language. A message baseline can (indirectly) be obtained through the use of a `timer` object. Objects of the `timer` class can be created, `reset` and `sample`'d; in the latter case it returns the time elapsed since it was last reset (or the baseline time of

```
unless e w d1 d2 = class
  tmr = new timer
  abortT := []

  start = before w + d1 action
    after w + d1 send accept

  drop = before d1 action
    abortT := (<- tmr.sample) : abortT

  accept = before d2 action
    t = (<-tmr.sample) - w - d1
    if elem True (map (inW t) abortT) then
      send e.drop
    else
      send e.accept
    abortT := (filter (inWOrL t) abortT)
    where
      inW t at = (at > t) && (at < t + w)
      inWOrL t at = at > t - d2

  result Cep {..}

struct Cep =
  start  :: Action
  accept :: Action
  drop   :: Action
```

Fig. 3. The `unless` class constructor.

creation). The built-in type implements arithmetic operations and comparison. The state variable `abortT` holds a list of baselines of occurred abort messages.

The `drop` method (invoked by an abort event) samples the current baseline (`<-` forces a synchronous execution of `tmr.sample`) and appends the result to the list (Timber uses a Haskell like list constructor) and updates the state:

```
abortT := (<- tmr.sample) : abortT
```

The `drop` method is given the explicit deadline `d1`, hence an abort message occurring at time $t$ will be processed before $t$+`d1`.

Let us now turn to the `start` method. Its only purpose is to trigger the `accept` method after `w+d1`, see Figure 4. In order to do so, it should be executed before `w+d1` (hence the explicit deadline). Notice, that there can be multiple outstanding `accept` messages defining (potentially) overlapping windows `w`.

The `accept` method operates under the explicit deadline `d2`, hence a triggering event `start` will be processed within `w+d1+d2`. First we obtain the start time of `w` through sampling the baseline and subtract window size `w` and the offset `d1`.

```
t = (<-tmr.sample) - w - d1
```

We then check whether the list of abort times holds any element which falls into `w`. Notice, we do this check after `w+d1`, hence we know that any abort messaged within `w` has been recorded. If the check is positive the alarm can be dropped (`send e.drop`) else raised (`send e.accept`). Finally, in order to free memory, the list of abort baselines is updated, filtering (retaining) only those which might abort outstanding `accept` messages; due to non-deterministic scheduling the earliest outstanding `accept` message window may start `d2` earlier than the current window.

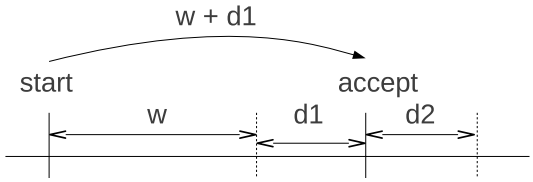



Fig. 4. Timing specification for `unless`.

As in the example, timing properties can be explicitly defined for the queries using parametrised class definitions in Timber. Subqueries must specify as tight (or tighter) timing constraints as its parent, e.g., the `ANY` query (and its subqueries) may operate under `w+d1` (derived from `start` in Figure 3), while the abort button must operate under the tighter deadline `d1` (derived from the timing specification for `drop` in Figure 3).

## III. Distributed Real-Time CEP

The overall architecture is given by the combination of:

- *distributed time synchronisation:* in order to get common time base within the network (or subnetwork),
- *real-time event based communication:* in order to transport events (with time-stamps) within bound time, and
- *real-time event processing under jitters and delays:* in order to deliver accurate query results within bound time.

In a distributed setting a query can be partially processed at different interconnected nodes. For the discussion we undertake the following set of assumptions:

- nodes are loosely coupled, hence define unique timing domains,
- communication links introduce a bound delay, and
- devices operate as specified, i.e., sensors readings, computations and transmissions are free of errors, and meet their timing requirements.

As discussed in Section II-A the problem of time synchronisation in loosely coupled systems is well known, and several approaches exists to mitigate timing errors and their effects. In general, the timing error (relative to what we consider accurate time) can be given as $E = e \pm \delta$, where $e$ is the steady state error and $\delta$ is the jitter. Assuming that the time synchronisation algorithms performs filtering over time, $e$ can typically be completely mitigated (minimised to 0), leaving us with the error $E = \pm\delta$.

In a loosely coupled system, communication can in general introduce both errors and (potentially) unbound delays. However, mechanisms for real-time communication have been developed providing bound delays, while transmission errors lead to limited resends, and ultimately packet loss.

In the proposed architecture, events are encoded in light weight SOA (CoAP/EXI) messages providing both bandwidth and computationally efficient transmission. On the receiver

side the SOA messages are turned into native Timber messages, given a permissible execution window relative to the time-stamp.

Figure 5 depicts a typical scenario, where sensors for *High_temp*[1] and *High_press*[2] are connected to a node b) which process the ANY[3] query, the sensor *Button_pressed*[4] is connected to node c) and the top level query UNLESS[5] is processed on node a). For the discussion, we consider the topmost node to hold the reference time a), while other nodes b) and c) may deviate from the reference time in terms of jitter $\delta^3, \delta^4$. We assume the communication links d) and e) to introduce bound delays $d(d), d(e)$.

While synchronisation errors and event delays are both common properties to loosely coupled systems, their effects with respect to CEP are largely different.

*1) Synchronisation Errors:* Synchronisation errors give rise to incorrect perception of the time an event occurred (time-stamp) which affects the matching process and may render both false positive and false negative query matches.

Consider the two events *High_temp*[1] and *Button_pressed*[4] , occurring at actual times $t_1$ and $t_2$. Assume $0 < t_2 - t_1 < w$ (i.e., that the alarm should be triggered and aborted). However, these atomic events are exposed to jitter for the time-stamping $\delta^3$ (*High_temp*[1]) and $\delta^4$ (*Button_pressed*[4]). Iff $t_2 - \delta_i^4 - (t_1 + \delta_i^3) \leq 0$ or $t_2 + \delta_i^4 - (t_1 - \delta_i^3) \geq w$ the abort event is missed and we falsely trigger an alarm, ($\delta_i^3$ and $\delta_i^4$ are jitter instances bound by $\delta^3$ and $\delta^4$ respectively). Assume $t_2 - t_1 < 0$ (i.e., the $abort$ event occurs before the $star$t event); iff $0 \leq (t_1 + \delta_i^3) - (t_2 - \delta_i^4) \leq w$ we falsely detect an abort event. Assume $t_2 - t_1 > w$ (i.e., the $abort$ event occurs after the $start$ event but outside the window $w$). Iff $0 \leq t_2 - \delta_i^4 - (t_1 + \delta_i^3) \leq w$ we falsely detect an abort event.

### A. Event Delays

In a distributed setting, events (representing subqueries) will be exposed to accumulated processing and communication delays for the subquery. However, as discussed in Section II-C the CRO based real-time CEP implementation offers specifying timing constraints for the processing of queries (and subqueries). Checking for schedulability (e.g., through response time analysis) involves the accumulated delays of the incoming events. However, schedulability analysis is out of scope for this presentation, in the following we simply assume that each subsystem is schedulable, (i.e, the timing specification for each real-time CEP query is met at each node).

Undertaking the assumption of successful time-bound communication, the accumulated delay for each subquery can be computed, thus we can reason on end-to-end delays for query processing. In the working example the ANY event will be delayed $d(d)+p(ANY)$ (where $p(ANY)$ is the time for processing of ANY and the triggering atomic event). Correspondingly *Button_pressed* will be delayed $d(e) + p(Button\_pressed)$.

Based on the timing specification for the working example unless, incoming ANY event operates under the timing constraint of method start, i.e., $d(d) + p(ANY) < w + d1$,

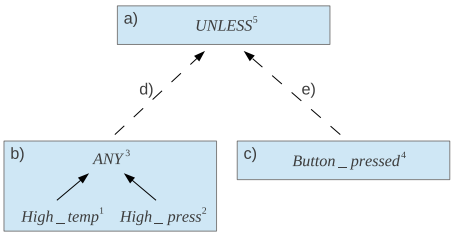



Fig. 5. Distributed CEP query.

and the *Button_pressed* event under the constrain of drop, i.e., $d(e) + p(Button\_pressed) < d1$. In this way bound processing and communication delays merely affect the timing performance of the query processing, not the correctness of the result. However, increasing $d1$ (to accommodate for larger delays), will render a larger window for retaining abort message baselines, thus lead to increased memory consumption.

### B. Safe Potential and Certain Matches

In this section we propose an extension to the CEP query definition and present a CRO based implementation that allows to manage time synchronisation errors in a safe way. To this end, we identify two distinct cases, *potential-* and *certain* matches, where the former matches all cases which cannot be ruled out (i,.e potential matches), while the latter triggers only on the set of cases where we can guarantee that a match has occurred. In this way the query programmer is given control as to how timing synchronisation errors (jitter) should be handled. In the working example, this amounts to matching a potential, certain or uncertain alarm situation (allowing to take appropriate actions for each case).

The proposed extension carries over to the query language that should be adapted accordingly, e.g., through introducing a CEDR rule UNLESS_CERTAIN, or through additional parameters UNLESS(CERTIAN,...). Similarly for the CRO-based implementation there are numerous options, Figure 6 sketches possible implementation of unless_certain.

For the UNLESS_CERTAIN clause it means that an alarm message should be emitted in the case ANY($High\_temp^1, High\_press^2$)[3] has occurred and we can rule out that any abort event $Button\_pressed^4$ has occurred within the time $w$ from a triggering event.

To rule out abort messages we have to check the interval $t - \delta^3 - \delta^4 < at < t + w + \delta^3 + \delta^4$, where $at$ is the baseline of the $drop$ message. If no abort message is found within the interval we are certain that the alarm has not been aborted.

## IV. Related work

In [25] O'Keeffe gives an excellent overview of existing technologies for data acquisition and processing in sensor networks and presents a language for complex event processing (building on interval based time-stamping of [26]). The presented CEP approach can guarantee "no false positives" (similar to our certain definition). However, in contrast to [25] our CEP engine is built directly on a framework for reactive

```
unless_certain e w d1 j1 d2 j2 = class
  ...
  accept = before d2 action
    ...
    where
      inW t at = (at>t-j1-j2) && (at<t+w+j1+j2)
      inWOrL t at = at > t-d2-j1-j2

  result Cep {..}
```

Fig. 6. `unless_certain` class constructor, with jitter parameters. Figure 7 depicts the timing specification with jitter.

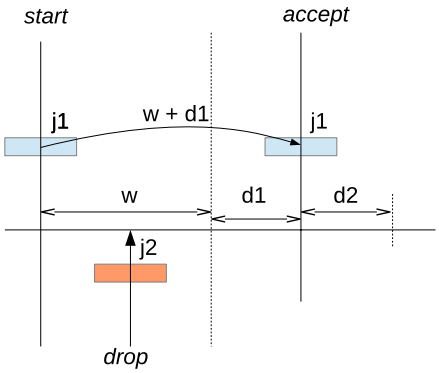



Fig. 7. Timing specification for `unless` with jitter (depicted by boxes).

(event driven) execution, which allows events to directly trigger arbitrary computations, state updates and I/O, under end-to-end timing in constraints (involving query processing). Other approaches involving probabilistic event processing (for timing, computation and communication errors) have been developed, e.g., [27]. Our approach is targeted towards distributed real-time applications applied onto lightweight nodes, hence computational and memory efficiency of the CEP implementation is a prime requirement, not feasible using AI approaches like [27]. Our approach also stands out by proposing the use of a stanards based SOA middleware, in contrast to proprietary middleware for the transportation and communication of (complex) events.

## V. Conclusions and Future Work

In this paper we have investigated the challenges involved to perform real-time CEP queries in a distributed setting. An architecture featuring state of the art methods for time synchronisation, event centric light weight SOA based communication, together with a real-time CEP mechanism has been presented. The impact of jitter and delay has been discussed and the real-time CEP mechanism has been extended to safely accommodate end-to-end timing requirements in a distributed CEP system. The taken approach does not try to hide the problems of jitter (due to time-stamping errors) and delay (due to processing and communication), but rather the opposite - we lift jitter and delay to the surface allowing the programmer/designer to take these into account already at design time. The approach, given correct assumptions on delay and jitter, is safe, while the actual execution (and output order of complex events) is non-deterministic. By *specifying* the timing requirements for the top level queries, the effect of non-determinism is under control. (Alternatively by *deriving* the timing requirements for the top level queries, the effect of non-determinism can be assessed and accounted for.)

As a proof of concept for the suggested approach, a prototype implementation has been devised. Taking the proof of concept and prototype implementation into a full scale demonstrator is ongoing work. The presented approach relies on assumptions of delay and jitter. Given worst case bounds, hard real-time guarantees are obtained. However, worst case delay and jitter can be hard to asses, and may often deviate vastly from the average behaviour. Hence for soft-real time applications, it could be of interest to investigate the statistical properties of the proposed CEP mechanism (given distributions rather than bounds for the parameters).

Currently the CoAP/EXI communication stacks are implemented external to the CRO model, hence run-time prioritising of processing is only applied to processing CEP queries. Migrating the communication stacks to Timber would allow for improved overall schedulability, and facilitate analysis at the node/device level. By distributing CEP queries, the load onto nodes and communication links may be balanced/optimised (in order, e.g., to reduce power consumption, minimise risk of contention, etc.). The optimisation of system partitioning under multiple criteria is an interesting topic for further investigation. This might be stretched to steer design choices affecting jitter/time synchronisation (e.g., NTP vs. PTP) and delay (media, protocols, network topology, etc.).

## Acknowledgments

This work is funded by the EU FP7 Project “AESOP”. The authors would like to thank partners of the AESOP project for discussions.

## References

[1] A. Colombo and S. Karnouskos, “Towards the factory of the future: A service-oriented cross-layer infrastructure,” in *ICT Shaping The World - A Scientific View, The European Telecommunications Standards Institute (ETSI)*. John Wiley and Sons Ed., April 2009, ch. 6.

[2] S. Karnouskos, A. Colombo, F. Jammes, J. Delsing, and T. Bangemann, “Towards an architecture for service-oriented process monitoring and control,” in *IECON 2010, (The 36th Annual Conference of the IEEE Industrial Electronics Society)*, 2010.

[3] G. Mühl, L. Fiege, and P. Pietzuch, *Distributed Event-Based Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[4] R. Kyusakov, J. Eliasson, J. Delsing, J. van Deventer, and J. Gustafsson, “Integration of wireless sensor and actuator nodes with it infrastructure using service-oriented architecture,” *IEEE Transactions on Industrial Informatics*, 2012, Accepted.

[5] R. T. Fielding and R. N. Taylor, “Principled design of the modern web architecture,” *ACM Transactions on Internet Technology*, vol. 2, no. 2, pp. 115–150, 2002.

[6] Z. Shelby, “Embedded web services,” *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 52 –57, 2010.

[7] *Efficient XML Interchange (EXI) Format 1.0*, W3C Std., March 2011. [Online]. Available: http://www.w3.org/TR/2011/REC-exi-20110310/

[8] M. Kovatsch, S. Mayer, and B. Ostermaier, “Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things,” in *Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012), Palermo, Italy*, 2012.

[9] GPS and Precision Timing Applications. (webpage) Last accessed 2012-12-20. [Online]. Available: http://www.cnssys.com/files/PTTI/hpan1272.pdf
[10] D. Mills, "Ntpv3 rfc 958. http://tools.ietf.org/html/rfc958," 1985.
[11] ——, "Internet time synchronization: the network time protocol," *Communications, IEEE Transactions on*, vol. 39, no. 10, pp. 1482 –1493, oct 1991.
[12] D. Mills, U. Delaware, and J. Martin, "Ntpv4 rfc 5905. http://www.ietf.org/rfc/rfc5905.txt," 1985.
[13] ——, "Ntpv4 rfc 5905. http://www.ietf.org/rfc/rfc5905.txt," 1985.
[14] R. Barga, H. Caituiro-Monge, T. Grust, H. Höpfner, A. Illarramendi, S. Jablonski, M. Mesiti, S. Müller, P.-L. Patranjan, K.-U. Sattler, M. Spiliopoulou, and J. Wijsen, *Event Correlation and Pattern Detection in CEDR*. Springer Berlin / Heidelberg, 2006, vol. 4254, pp. 919–930. [Online]. Available: http://dx.doi.org/10.1007/11896548_70
[15] R. S. Barga, J. Goldstein, M. H. Ali, and M. Hong, "Consistent streaming through time: A vision for event stream processing," in *CIDR*, 2007, pp. 363–374.
[16] P. F. N. S. Darko Anicic, Sebastian Rudolph, "Stream reasoning and complex event processing in etalis," *Semantic Web Journa:Special Issue: Semantic Web Tools and Systems*, 2012.
[17] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic, "Ep-sparql: a unified language for event processing and stream reasoning," in *Proceedings of the 20th international conference on World wide web*, ser. WWW '11. New York, NY, USA: ACM, 2011, pp. 635–644. [Online]. Available: http://doi.acm.org/10.1145/1963405.1963495
[18] M. Carlsson, J. Nordlander, and D. Kieburtz, "The semantic layers of Timber," in *First Asian Symp. on Programming Languages and Systems (APLAS)*, ser. Lecture Notes in Computer Science, vol. 2895. Berlin, Germany: Springer-Verlag, 2003, pp. 339–356.
[19] A. P. Black, M. Carlsson, M. P. Jones, R. Kieburtz, and J. Nordlander, "Timber: A programming language for real-time embedded systems," Oregon Graduate Institute School of Science & Engineering, Tech. Rep., 2002.
[20] The Timber Language. (webpage) Last accessed 2011-04-15. [Online]. Available: http://www.timber-lang.org
[21] J. Eriksson, "Embedded real-time software using TinyTimber : reactive objects in C," Licentiate Thesis, Luleå University of Technology, 2007.
[22] J. Wiklander, "A reactive approach to component-based design of resource-constrained embedded systems," PhD Thesis, Luleå University of Technology, 2011.
[23] P. Pietrzak, P. Lindgren, and H. Mäkitaavola, "Towards a lightweight CEP engine for embedded systems," in *The 38th Annual Conference of the IEEE Industrial Electronics Society, Montreal* , Oct. 2012.
[24] P. Lindgren, P. Pietrzak, and H. Mäkitaavola, "Real-Time Complex Event Processing using Concurrent Reactive Objects," in *IEEE International Conference on Industrial Technology (ICIT 2013)*, Oct. 2013, p. Accepted for publication.
[25] Dan OKeeffe, "Distributed Complex Event Detection for Pervasive Computing," University of Cambridge, Tech. Rep., 2010.
[26] C. Liebig, M. Cilia, and A. Buchmann, "Event composition in time-dependent distributed systems," in *Proceedings of the Fourth IECIS International Conference on Cooperative Information Systems*, ser. COOPIS '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 70–. [Online]. Available: http://dl.acm.org/citation.cfm?id=520790.793809
[27] A. Artikis, O. Etzion, Z. Feldman, and F. Fournier, "Event processing under uncertainty," in *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '12. New York, NY, USA: ACM, 2012, pp. 32–43. [Online]. Available: http://doi.acm.org/10.1145/2335484.2335488