Data Dissemination Supporting Complex Event Pattern Detection

R. BALDONI, S. BONOMI, G. LODI, M. PLATANIA, L. QUERZONI
Dipartimento di Ingegneria Informatica Automatica e Gestionale Antonio Ruberti
Sapienza Universitá di Roma
Via Ariosto 25, 00185 Rome, Italy

The increasing use of the Internet and the improvement in hardware technology led most of the application previously deployed in “closed” environments, such as business intelligence, smart environments, complex system software management, to federate into geographically-distributed systems. Such applications use “sense-and-respond” capabilities, i.e., they correlate basic events that could potentially occur at different sources and detect complex event patterns, in order to timely and properly react to changes that may happen within the system. In this context, a fundamental role is played by the data dissemination service that brings events from producers to consumers where complex event patterns are detected. In this paper we discuss the characteristics that a data dissemination service should have in order to support in the best way the complex event pattern detection functionality, and present an assessment of a number of technologies that can be used to disseminate data in the earlier mentioned context. We also describe how those technologies can be effectively deployed in scenarios where numerous independent data sources produce large amounts of events in the form of high-throughput streams. Finally, we present a matching between distributed application requirements and the capabilities offered by the data dissemination services used to implement them, highlighting which aspects should be considered in the design of novel middleware solutions to fill this gap.

Keywords: Data Dissemination, Event Pattern Detection, Event Processing, Collaborative Systems.

1. INTRODUCTION

Today’s modern enterprises look for novel IT services able to withstand continuously evolving business need to be pursued in strongly dynamic environments [NESSI 2009]. The backbone of these scenarios is a dynamic and loosely coupled distributed system formed by autonomous entities (e.g., nodes, processes, organization clouds) distributed across different administrative domains. Such entities employ “sense-and-respond” capabilities, required to augment their perceived knowledge of the global business scenario state and timely and appropriately respond to changes that may occur. The possibility to federate and cooperate offers the opportunity to pool resources together and share data for common benefit [Balazinska et al. 2004; Huang et al. 2007]. Entities can be composed so as to form complete end-to-end services such as business intelligence, smart environments, collaborative security, stock market, and thus pave the way to collaborative executable enterprises [NESSI 2009]. In order to provide such end-to-end services it is crucial to support *event-driven computing* so as to monitor and detect complex event patterns generated in nearly real-time.

Complex Event Pattern Detection (CEPD) is becoming a fundamental capability for a variety of monitoring applications. It consists of detecting complex event patterns that may occur over a certain time period and possibly within a certain spatial distance. In most of the existing CEPD systems (e.g., [Esper 2009; JBoss 2010]), distributed and possibly heterogeneous event sources originate simple, basic events that are continuously pushed to a CEPD module. The CEPD module uses *operators* to evaluate and recognize, with continuos queries and rules, the occurrence of specific complex patterns of events that could have temporal and/or spatial relationships.

In order to exploit all that information available at different sites, it is mandatory to employ a data dissemination service that allows both the large volume of basic events originated from

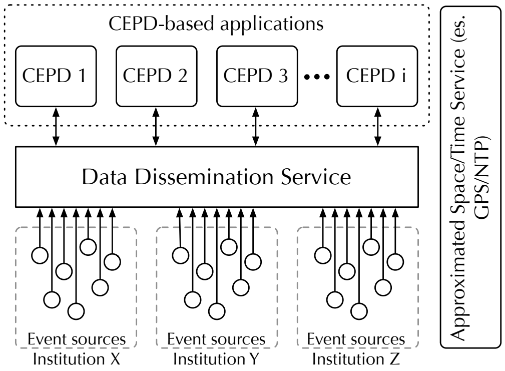

Figure. 1. The data dissemination service for CEPD-based applications.

those sites to reach the CEPD destination systems for processing, and the results of the CEPD computation to reach all the sites interested in receiving them. The data dissemination service should guarantee a number of Quality of Service (QoS) requirements (e.g., ordering, timeliness, reliability) in order to sustain high-throughput systems and the use of all operators in CEPD-based applications: without these guarantees, the systems could run the risk, for instance, to miss events that are decisive for detecting complex event patterns, or to deliver events and disseminate results with unpredictable delays thus loosing the potential benefits of the CEPD computation as interested receivers might be unable to timely react to what has been correlated and detected. In addition, if the data dissemination service does not provide ordering, events can arrive out-of-order to a CEPD module, preventing the possibility to apply temporal operators to recognize sequence patterns [Pietzuch et al. 2004]. As such, CEPD implementations [Esper 2011] are forced to provide their own reordering mechanism in addition to their native functionalities, to overcome the deficiencies of the data dissemination service.

The main contributions of this paper are thus the following:

- we present a model of a data dissemination service for collaborative event detection environments (Section 2 and Section 3);
- we review a number of technologies, available on the market, and well known data dissemination paradigms that can be used for the implementation of the data dissemination service (Section 4);
- we present several case studies highlighting requirements and how they can be addressed in their specific contexts (Section 5), and finally
- we individuate the main aspects that should be considered in the design of novel middleware solutions to fill the gap between the application requirements and the capabilities offered by the available data dissemination technologies (Section 6).

2. COMPLEX EVENT PATTERN DETECTION

Our reference system model is depicted in Figure 1. A possibly large set of sources sends streams of raw events; events are managed by a data dissemination service that brings events to CEPD modules which in turn derive complex events with higher level significance. Complex events are obtained by properly correlating basic events that are apparently uncorrelated. We consider that each CEPD module can be potentially interested in receiving any subsets of basic events and the CEPD correlation be based on the ability to recognize patterns of basic events, namely complex event patterns, that exhibit logical, timing and/or spatial relationships among them.

Event Sources. We assume event sources are loosely synchronized and geographically dispersed over different administrative domains. They can access a coarse grain common clock (provided for example by the Network Time Protocol (NTP) service) and might have knowledge about their geographical location. This means that events can be labeled with timing and geographical timestamps. Note that due to the typical unpredictable delay of loosely coupled distributed systems like the Internet, this kind of synchronization cannot be used to reliably and totally order events produced by independent sources [Liebig et al. 1999]. In this discussion we assume that the aggregated traffic generated by the sources can reach hundreds or thousands of events per second which entails that the data dissemination service has to sustain high throughput.

CEPD Module. As in [Pietzuch 2004], a CEPD module can be a finite state automaton which might be able to detect several patterns concurrently. Operators are used to express these patterns and can be embedded in a programming language; the automaton is then obtained by compiling the language. We assume that a CEPD module can ideally keep in memory all the events received from the data dissemination service, i.e., event patterns cannot be missed by the fact that the CEPD module drops events due to memory limitations. Another important aspect for a CEPD module is when the automaton can consume an event (i.e., the detection policy) coming from the data dissemination service. In the following we discuss pattern operators and detection policies respectively.

Event Pattern Operators. Usually operators work on a set of events C, namely the context, kept in memory by the CEPD module (e.g., C could be the entire set of events kept in memory by a CEPD). To express complex patterns, operators can be broadly grouped into the following five classes:

- *Logical operators:* logical pattern operators are `and`, `or`, and `not`. As an example the pattern `e and e'` is detected as soon as e and e' belongs to C.

- *Quantifier operators:* given a predicate $P(e)$, the `any` operator states that any event e in C makes $P(e)$ true; the `exists` operator states that there is at least one event e in C that makes P true.

- *Temporal operators:* these operators are related to the time interval in which events occur. They include `sequence` (identified by a list of events) stating that the pattern is satisfied if there exists a sequence of events occurred in the same order of the pattern definition list, and a number of operators such as `time interval` and `time within` [Esper 2009] that can denote the time interval and the time instant the events occurred.

- *Counting operators:* counting operators include, among others, `count` which counts how many times certain basic events or event patterns have been detected into the context C [Etzion 2008].

- *Spatial operators:* these operators include `distance` used to evaluate if the events occur within a certain distance, and `moving` used in case the events describe consistent movements to a certain direction [Etzion 2008]. The spatial operators can be conveniently used in case of processing of events for instance generated by sensors in smart houses [Pietzuch 2004].

Note that all these operators can be used in conjunction, thus producing complex pattern expressions. For example, using the syntax of the open source complex event processing engine Esper [Esper 2009], the pattern `(A or B) where timer:within (5000)` matches for any A or B events in the next 5 seconds. More complex operators can be also devised, for example based on statistical property of the context (e.g., trends).

Detection Policy in CEPD. An important part of the design of a CEPD module is to determine the right time to consume an event coming from the data dissemination service. As remarked by Pietzuch in [Pietzuch 2004], the problem is to decide when the next event in the event input stream can be safely consumed by the automaton without running the risk that an event with an older timestamp is still being delayed by the network. Premature consumption could lead to incorrect detection or non-detection of an event pattern. Pietzuch identifies two main detection policies:

• *Best Effort Detection* (BED): basic events are consumed as soon as they arrive to the CEPD module. This policy may cause incorrect detection and then it can be applied by applications sensitive to delay and not to false positives.

• *Guaranteed Detection* (GD): basic events are consumed when all the preceding events are available (i.e. basic events can be consumed when they are *stable*). This policy requires that basic events have to be delivered by the data dissemination service respecting some ordering property and avoiding the CEPD module to ignore patterns that should be detected. In an asynchronous setting, the guaranteed policy can introduce an unbounded delay. To avoid this problem, a *Probabilistic Stability* policy can be used to allow the CEPD module to consume events when they are stable according to a specified probability. This latter policy is a trade-off between BED and GD that makes possible to relax the constraint on the event deliveries of the data dissemination service.

Application-specific Timed Detection. The detection of a complex pattern of events is an activity that can possibly happen at any point in time after the events that constitute the pattern have been generated by (multiple) sources. However, from an application point of view, it is often desirable that a pattern that happened at the sources is recognized as quickly as possible by the CEPD module, that is, within some application dependent time bound. As an example, a collaborative security application could require that detection should happen within a maximum of 10 seconds after the occurrence of the patter at the sources. The detection of a pattern can happen only when all the events that constitute the pattern reached the CEPD module and the module itself has completed the elaboration. Therefore, if both of these activities are guaranteed to end within the application time bound from the production of the last event that constitutes the pattern, the system is able to provide a *timed detection* service for a given application. Note that if the service is not timed, the detection of patterns can become useless from an application viewpoint as interested receivers might be unable to timely react to what has been correlated and detected.

3. DATA DISSEMINATION SERVICE

A data Dissemination Service (DS) for event pattern detection aims at routing each event from its source to all the CEPD modules interested in receiving it. The DS is characterized by its functional and non-functional properties.

From a functional point of view, all different DSs can sport distinct event selection models: *no-selection*, *channel-based* or *content-based*. The no-selection model captures the typical behaviour of middleware products offering broadcast functionalities: participants simply join the system and receive all the messages broadcasted in it. The channel-based model, together with its close sibling, the *topic-based* model, assume that every event injected in the system is completely characterized by a name, representing the channel or topic it is published in; processes interested in receiving events can declare their interest in just a subset of all the published information by joining the channels where this events are expected to be injected. The content-based model is the most general as it assume all events are characterized by a set of attributes (with their corresponding values); potential receivers can issue complex subscriptions by applying constraints on available attributes such that the DS will deliver them only events satisfying their requirements. Depending on the event-selection model, the DS is therefore able to perform pre-filtering of events destined to a CEPD module and thus to reduce the overall load the module will need to sustain.

Non-functional aspects characterizing the DS can be grouped on three categories: reliability, ordering and timeliness properties.

Reliability properties define which kind of guarantees are provided by the DS in delivering each single event to its intended destinations. We can distinguish between two different properties:

- *Best-effort delivery:* the DS will provide the event routing service without any specific guarantees on the delivery; from this point of view, receivers can expect to miss some events that will not be delivered due to unexpected causes (e.g. message losses in the underlying transport layer); DSs providing best-effort delivery are usually designed to offer the best performance (obtained by avoiding as much as possible any protocol overhead); despite the lack of specific mechanisms to enforce stronger delivery properties, such systems usually behave in a good way as long as they are deployed on top of a fairly reliable communication substrate.

- *Reliable delivery:* the DS will guarantee the delivery of events to all their intended destinations despite possible network losses or other unexpected events; this property is usually enforced at the event routing level through several different techniques like retransmission, broker replication, multi-path delivery, etc. The overhead caused by the property enforcement can have a non-negligible impact on the overall performance.

Note that the enforcement of a reliable delivery property implies a strict definition in the DS of the set of events that must be delivered to each destination. This becomes a non-trivial issue as soon as you consider that destination process can possibly change at runtime their subscriptions and that the interactions taking place between event producers and consumers are completely asynchronous and decoupled with the purpose of improving system scalability [Baldoni et al. 2005].

Ordering properties provide a mean to define a specific order that must be enforced by the DS when delivering events to consumers. Several different, and partially orthogonal ordering properties can be considered:

- *Per-source FIFO order:* this is the simplest form of order that can be considered as it forces the DS to deliver events published by a same producer in the same order as they were produced (i.e. with a FIFO semantics); as a consequence, the delivery of events generated by distinct sources can experience different interleaving on distinct destinations.

- *Causal order:* by enforcing this property, the DS guarantees that if a process publishes an event e' after an event e has been delivered to it, then the DS will not deliver to a second process the events in the order e', e; this property proves particularly useful in those context where processes act both as producers and consumers and where maintaining the causal relationship among multiple events is fundamental for the application correctness.

- *Total order:* by enforcing this property, the DS guarantees that any two destinations that receive a same set of events will receive those events exactly in the same order; this order not necessarily has a connection with the real publishing time instants of these events, i.e. two events e and e' published at time t and t', with $t < t'$ can be delivered either in the order t, t' or t', t, but not a mix of the two.

- *Real-time order:* this property closely resembles the total order property but also require deliveries to be executed in the same order as events have been produced (hence the *real-time* attribute); the enforcement of this property can be usually guaranteed only with a tolerance due to the impossibility to perfectly synchronize event sources and thus always correctly order the publication time of concurrent events.

The real-time order property is clearly the most comprehensive as it subsumes all the preceding ones. Causal and total order are orthogonal and can be enforced together, if needed.

Each of these properties, if satisfied by the data dissemination service, can determine if a given detection policy can be used for a certain type of event pattern. Let us consider a data dissemination service that does not guarantee the Reliability property. In this case, any event can be lost or delivered only to a subset of the interested CEPD modules; as a consequence, independently of the operators describing the pattern, it would be possible to ensure only a BED policy and not a GD policy. In other words, the Reliability property is a necessary condition for adopting a GD policy.

Ordering properties are not needed to deterministically detect most of the patterns (i.e. it is not needed to adopt a GD policy): logical, quantifier, counting, time interval, time within and spatial operators can be detected deterministically even if the data dissemination service guarantees reliable delivery and does not satisfy any order as these operators consider just the set of occurred events and not the order in which those events took place. In contrast, in order to deterministically detect patterns involving the sequence operator, the total (or real-time) order property is mandatory, otherwise only a BED policy detection can be used. Needless to say, if both reliability and total ordering are satisfied the GD policy can be effectively used and all the event patterns can be deterministically detected.

Timeliness express the ability of the DS to provide the expected service within known time bounds [Baldoni et al. 2003]. This aspect proves crucial in many mission-critical applications, and it directly impacts the time needed by the DS to route an event from its publication point up to all its intended destinations.

- *Timely delivery:* there exists a time interval Δ such that, given any event e delivered at a CEPD module at time t, e has been published at a time t' where $t - \Delta \leq t' < t$.

The timely delivery property does not directly impact the kinds of patterns that can be detected by the CEPD modules and it does not impact the detection policies. However, the time needed for the data dissemination service to convey events to the CEPD module has a strong influence on the timed detection; if the application time bound is larger than the value of Δ, no timed detection can be achieved. Note that, in the timeliness property, the value of Δ can also depend upon the throughput that the DS has to sustain: the higher is the throughput, the larger is the value of Δ; this, in turn, influences the timed event pattern detection.

4. TECHNOLOGIES FOR DATA DISSEMINATION

This section reviews a number of state of the art data dissemination technologies and approaches that can be considered for supporting CEPD systems. We distinguish between technologies available on the market and well-known paradigms that can be used to implement a DS.

4.1 Data Distribution Service (DDS)

The OMG's Data Distribution Service for Real-time Systems (DDS) is an API specification and interoperability wire-protocol that defines a data-centric publish-subscribe interaction paradigm [Corsaro et al. 2006]. DDS is based on a fully decentralized architecture, which provides an extremely rich set of configurable QoS policies to be associated with topics. A publisher can declare the intent of generating data with an associated QoS and writing the data in a topic. The DDS is then responsible for disseminating data (in either a reliable or best-effort fashion) in agreement with the declared QoS, that has to be compatible with the one defined by the topic. Data Distribution Service also offers a high level of subscription expressiveness, by allowing subscribers to declare filters on the content of events published on their topics of interest.

The DDS provides a set of QoS policies in order to control the timeliness properties of distributed data. Specifically, it defines the maximum inter-arrival time for data and the maximum amount of time that should elapse for distribution of data from publishers to subscribers.

Owing to the properties discussed in Section 3, DDS guarantees reliability and timeliness in the data dissemination. However, no total ordering reflecting real-time event generation is ensured for events originated from multiple and heterogeneous sources: the DDS guarantees just a per-source FIFO order. In addition, its guaranteed QoS properties can be effectively applied only when the DDS is deployed in a strictly controlled setting (i.e., in a managed environment); in a large scale, unreliable and unmanaged context as the collaborative event detection environments can be, the performance obtainable by the DDS may become unpredictable [Baldoni et al. 2008], thus compromising the possibility to support high throughput CEPD systems.

4.2 Java Message Service (JMS)

The Java Message Service (JMS) [Microsystem 2008] is a standard promoted by Sun Microsystems to define a Java API for the implementation of a topic-based message-oriented middleware. A JMS implementation represents a general-purpose *Message Oriented Middleware* (MOM) that acts as an intermediary between heterogeneous applications: the applications can choose the communication mode that better suits their specific needs such as pub/sub and point-to-point modes.

JMS allows an application to require every message to be received once and only once or choose a more permissive (and generally more efficient) policy, which permits to drop and duplicate messages. It supports various degree of reliability through different basic and advanced mechanisms. Basic mechanisms include: *message persistence* through which a JMS application can specify that messages are persistent, *message priority levels* through which an application can define urgent messages, and, finally, *message expiration* through which an application can set a message expiration time in order to prevent duplicated messages. The most advanced mechanism consists in the creation of durable subscriptions that allow subscribers that are idle to receive messages as soon as they come back on-line. Other common features in MOM products, such as load balancing, resource usage control, and timeliness of messages, are not explicitly addressed in the JMS specification.

With respect to the properties discussed in Section 3, JMS guarantees reliability through different mechanisms including message persistence. However, as earlier stated, no timeliness and total order reflecting the real-time event generation can be provided. As DDS, even JMS guarantees a per-source FIFO order. Finally, it is worth noticing that JMS is typically deployed through the use of a central server that implements all the MOM functionalities. This solution can then suffer from inherent drawbacks of a centralized system. The central server can become a single point of failure or security vulnerability: if the server crashes or is compromised by a security attack, the data dissemination process can be jeopardized. In addition, the volume of events the central server can disseminate in the time unit is limited by the server's processing and bandwidth capacities which prevent the system to be sufficiently scalable to support high throughput CEPD systems.

4.3 Multicast-based Solutions

In the nineties there has been a large body of research on multicast platforms for building reliable and consistent distributed systems (e.g., [Birman 1993]). This research has produced interesting add-on to commercial products (e.g., [JGroups 2010]) that are focused to keep consistent a certain number of replicas. One of the main consistency models introduced for such platforms has been the Virtual Synchrony. On top of that, several different types of multicast primitives have been proposed for ordered and reliable multicast diffusion (causal multicast, total order multicast) [Birman and Joseph 1987]. Data dissemination could be implemented by using such platforms that would deliver in a consistent and ordered way events to CEPD modules. Different implementations can consider either a single group of destinations that receive all messages (non interesting messages are filtered out at application level), or several groups populated by nodes interested in the same topic [JGroups 2010]. However, it is well known that these platforms work for small number of groups of small sizes. In most implementations, performance does not scale in terms of either number of groups, large groups or high multicast rate and they can show instability. This lack of scalability on several dimensions prevents the usage of such technology in a collaborative large scale environment. Currently, there is an interesting field of research that aims at providing scalable multicast platforms in the context of cloud computing. This research could reconcile some degree of consistency of multicast with the need to sustain high throughput (e.g., [Vigfusson et al. 2010]).

4.4 Gossip-based Solutions

The recent shift from small/mid-scale distributed systems deployed on very controlled environments, to large or huge-scale systems, geographically distributed over the world where processes interact using unreliable links that traverse several independent administrative domains showed the limits of traditional deterministic approaches to information dissemination. This shift led to the design of novel data dissemination algorithms based on the gossip paradigm. These algorithms are based on the so-called *epidemic approach* where data is disseminated like the spread of a contagious disease or the diffusion of a rumor. This approach has several advantages that have been thoroughly studied: few initial infection points are sufficient to quickly infect the whole population as the number of infected processes grows with an exponential trend. Moreover, these algorithms are also strongly resilient to the premature departure of several processes, making them very robust against failures. The gossip approach has been successfully applied to a variety of application domains such as database replication [Demers et al. 1987], cooperative attack detection [Zhang and Parashar 2010], resource monitoring [Van Renesse et al. 2003], and publish/subscribe based data dissemination [Costa and Picco 2005]. With respect to the event selection models defined in Section 3, gossip protocols can sport either the *no selection*, or the *channel-* or *content-based* model, as in [Costa and Picco 2005].

Taking into account the properties of an ideal data dissemination service, most of such algorithms based on the gossip paradigm are able to deliver a huge amount of events in a geographically distributed setting with nice reliability properties. Thanks to the quick spread of "infections" also the time figures are very interesting. However, such properties can be guaranteed only on a probabilistic basis, thus allowing only best effort policies. Moreover, gossip-based algorithms usually do not ensure any of the previously defined order properties.

5. CASE STUDIES

In this section we discuss different application domains in which CEPD modules are used and for which the data dissemination service has to guarantee specific Quality of Service (QoS) requirements in order to sustain CEPD computations.

5.1 Collaborative Security for Financial Critical Infrastructures Protection

Financial institutions are increasingly exposed to a variety of security related risks, such as massive and coordinated cyber attacks [Deutzman 2010; Vijayan 2004] aiming at capturing high value (or, otherwise, sensitive) information, or disrupting the service operation for various purposes. Single financial institutions use local tools to protect themselves from those attacks (e.g. intrusion detection systems, firewalls); these tools verify whether there exists some host that performs suspicious activities within certain time windows. However, due to the complexity of today's attacks, such kind of defense results inadequate. A more large view of what is happening at all financial institution sites is required, that could be obtained by *collaboratively* sharing and correlating the information coming from them, thus improving chances of identifying low volume activities which would have gone undetected if individual institutions were exclusively relying on their local protection systems [Lodi et al. 2009].

Figure 2 illustrates the scenario of collaborative protection of financial critical infrastructures against inter-domain stealthy SYN port scan attacks[1]. The attack is a form of port scan that aims at uncovering the status of certain TCP ports without being traced by application level

[1] This scenario has been widely investigated in the context of the EU project CoMiFin [CoMiFin 2008]

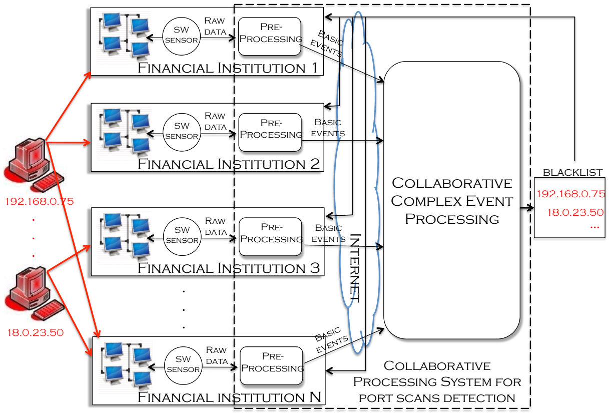

Figure. 2. Collaborative processing system for port scan detection in financial critical infrastructures.

loggers. It is carried out by probing a few ports of interest at different financial institutions, so as to circumvent configured thresholds of local protection systems, and delaying those probes in order to also bypass local time window controls [Lodi et al. 2011; Aniello et al. 2011]. A scanner targeting different financial institutions probes ports on different targets by following a well-known data pattern. For instance, a common pattern in an inter-domain stealthy port scan is to initiate so-called *incomplete connections*; that is, TCP connections for which the three-way handshaking consists of the following ordered sequence of packets: SYN → SYN-ACK → RST[2] (or nothing after a timeout is expired).

Owing to this scenario, a possible collaborative processing system can be built that takes in input different basic events representing part of the traffic sniffed from financial sites' networks. It is worth noticing that basic events are obtained through specific pre-processing activities carried out locally by financial sites (see Figure 2). These activities allow the sites to control the data flow to be injected into the collaborative processing system: pre-filtering and possibly sensitive data anonymization operations are thus performed in this phase. Hence, from a functional point of view, a simple *non-selection* model of the data dissemination service can be employed.

Basic events flow from multiple financial sources to one or more CEPD modules for collaborative processing purposes. The CEPD modules verify the presence of the earlier mentioned data pattern and detect whether that pattern is "frequently" discovered from all the sites (i.e., CEPD modules verify if the total number of collaboratively detected incomplete connections exceeds a pre-defined threshold). To this end, CEPD modules can use all the operators discussed in Section 2 (with the exception of the spatial ones). At the end of the processing, if a high number of malicious activities originated by specific IP addresses are observed, a blacklist containing those addresses is produced and disseminated to all the sites of the system (see Figure 2).

In the design of such an architecture, it clearly emerges the crucial role played by the data dissemination service. The service is required in order to allow both large volume of basic events to reach the collaborative processing system, and the results of the processing to reach all the sites interested in receiving the produced blacklist.

However, in order to be effective, data dissemination has to guarantee that specific properties among those identified in Section 3 are enabled in order not to compromise the port scan detection capabilities of the overall collaborative processing system.

Required data dissemination properties. For an accurate detection of inter-domain stealthy port scans, data dissemination should provide the following non-functional aspects:

- *reliable delivery* of both the basic events to the collaborative CEPD modules and the produced

[2] the "→" represents the sequence operator in the well-know CEP engine ESPER [Esper 2009].

blacklist to financial sites of the system. Referring to the above data pattern, if the SYN-ACK packet is lost in the communication between the sources and CEPD modules, the pattern cannot be correctly detected. Then, the pattern will not contribute to the computation of the earlier discussed pre-defined threshold, thus augmenting the number of false negatives (i.e., real scans that are not detected) and decreasing the detection accuracy of the system.

• *per source FIFO order* of the basic events. Referring to the above data pattern, if some of those packets are delivered out of order at the level of single source, the CEPD modules will not recognize the correct sequence. This entails that even if the data pattern has been carried out by some malicious IP addresses, it cannot be detected and used in the general threshold computation. Similarly to the previous point, this may lead to augment the number of false negatives and, then, to decrease the port scan detection accuracy of the system.

• *timely delivery* of both the basic events and produced blacklist. Typically, these types of attacks are characterized by very low execution and propagation times, in the order of a few seconds depending on the number of target hosts and ports on those hosts. An effective processing system should then detect scanner IP addresses and disseminate the results to interested sites within that application time bound. This allows the sites to exploit the intelligence produced by the collaborative system and to timely take proper countermeasures. In doing so, the data dissemination has to guarantee timeliness on both the delivery of basic events and the dissemination of produced results. In particular, we claim that in the inter-domain stealthy port scan the final result of the collaborative computation should be delivered within a time interval which is comparable to the application time bound (i.e., within a few seconds).

The scenario of Figure 2 has been implemented in two different versions: one version makes use of JMS as data dissemination service [Lodi et al. 2011], and the second version uses multicast technologies [Aniello et al. 2011]. In both cases, the different types of technologies were well-suitable for what concerns reliability and ordering properties, as required by the above application domain (in the case of JMS, reliability can be guaranteed provided that the central JMS server does not crash). As for the timeliness, the system was able to sustain high throughput processing in a timely fashion (in the order of seconds) only in small groups of participating organizations and within high bandwidth environments. In a large scale scenario with an increased number of participants and decreased available bandwidth connecting the financial sites to the collaborative processing system, as emerged from the assessment discussed in the previous section, the timeliness requirement was no longer met [Aniello et al. 2011].

5.2 Algorithmic Trading for Stock Market

The huge improvement in hardware and communication technology led the stock market to extend its services from physical locations, where buyers and sellers meet and negotiate, towards a virtual market place (NASDAQ, NYSE Arca and Globex, to cite a few) that exploits electronics media, where traders can transact from remote locations. The increase of the electronic trading brought several benefits, such as reduced cost of transactions, greater liquidity and competition (allowing different companies to trade with other ones) and increased price transparency [Rao 2010].

On the contrary, the virtual market place poses unprecedented challenges in terms of data flows (up to 1 million updates per second) and short-term trading decisions (down to a thousand of a second). These problems are further exacerbated by the complexity of the operations. As an example, consider the following indication about when to buy or sell a stock: *"When the price of IBM is 0.5% higher than its average price in the last 30 seconds, buy 10000 shares of Microsoft every 3 seconds unless the average price drops back below the same threshold"* [Palmer and Dzmuran 2009]. Such an indication requires a careful market analysis of the last IBM and Microsoft's share prices and a consequent decision whether to buy or not these shares. The complexity of operations and the strict timeliness constraints imposed by the electronic market, led to the use of sophisticated algorithms that process event streams, make complicated calculations and take intelligent decisions in response to changing conditions reflected in those

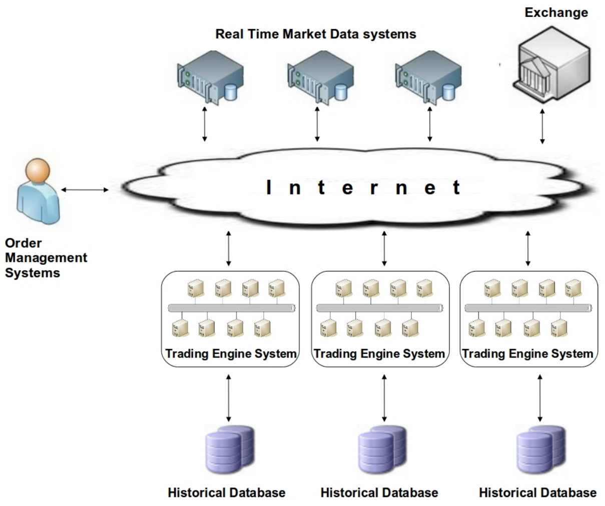

Figure. 3. Virtual market place with a CEPD system that implements an algorithmic trading engine.

events. This strategy, called *algorithmic trading*, is becoming more and more a fundamental component of the virtual market place: all the worlds top-tier firms, including JP Morgan, Deutsche Bank, and ABN Amro, as well as buy-side hedge funds, such as Aspect Capital, are applying algorithmic trading.

Due to its inherently ability to process data streams, to correlate events and to recognize complex patterns in a timely fashion, Complex Event Processing is at the core of algorithmic trading engine implementation [Palmer and Dzmuran 2009]. Many of the firms cited above are using event processing as their technical architecture for algorithmic trading. A typical virtual market place, where a system of CEPD engines is used to infer when to buy or sell shares, is depicted in Figure 3. Each firm in the electronic market has its Trading Engine System: it is composed by several CEPD engines that take in input client orders stored in the Order Management System. CEPD engines can obtain market data directly from the Exchange or from the Real Time Market Data. The Market Data sends information such as the current pricing and the number of contracts, as they are provided in real time by the Exchange. Examples of the Market Data are Reuters, Bloomberg and Wombat. In addition, the Trading Engine System can always purchase historical market data directly from the market's Exchange. CEPD engines execute complex queries on the data according to the instruction provided by clients. To this end, they use *Logical, Quantifier, Temporal* and *Counting* operators as defined in Section 2. In addition, CEPD engines can be organized in a hierarchical manner, with some of them that operate on raw data and produce more complex events that will be analyzed by engines at a higher level in the hierarchy. The outcome of the executions and the order status is sent back to the Order Management System. Finally, raw and/or summarized historical data can also be stored on the Historical Database and utilized for future strategy decision.

The communication among the electronic market components flows through the Internet. Because of the strict requirements imposed by stock market applications, the data dissemination service plays a key role in the implementation of such system. In particular, it is required to convey a large volume of raw events from the Order Management System and the Exchange to the Trading Engine System in a reliable and timely fashion, in order to let CEPD engines properly apply their strategy.

Required data dissemination properties. With respect to the properties identified in Section 3, the data dissemination service for algorithmic trading in stock market has to provide:

- *reliable delivery:* raw data produced by the Exchange or the Real Time Market Data and complex events generated by CEPD engines, as well as client orders, outcome of operations and order status feedback, must be reliably delivered to all interested destinations. Indeed, the loss of a message can compromise the accuracy of the applied strategy. As an example, consider the complex query described above, with a trading algorithm that has to buy 10000 Microsoft's shares when the price of IBM is 0.5% higher than its average price in the last 30 seconds. If the event *"the price of IBM is 0.5% higher than its average price"* occurred in the last 30 seconds is lost, the algorithm does not buy the Microsoft's shares, preventing the client from a possible profit.

• *timely delivery:* it is a strict requirement of the stock market applications, because the delay in the delivery of an event can lead the Trading Engine System to take a wrong decision. Consider, again, the previous example: if the event *"the price of IBM is 0.5% higher than its average price"* occurred in the last 30 seconds is notified to CEPD engines with a delay that exceeds the value Δ imposed by the application, then the algorithm might not buy the Microsoft's shares, preventing, even in this case, the client from a possible profit. In addition, the timeliness in the information delivery must be ensured not only for row events produced by the Exchange or the Real Time Market Data, but also for complex events generated by CEPD engines that are input for other engines at a higher level of the hierarchy.

• *real time order:* the information produced by the Exchange or the Real Time Market Data must be delivered in a real time order to all Trading Engine Systems. Such a requirement is fundamental to ensure a fair electronic market service to all clients. As an example, consider the three events: e_1 = *"the price of IBM is 0.5% higher than its average price"*, e_2 = *"the price of Apple is 0.8% lower than the price of IBM"* and e_3 = *"buy 10000 Microsoft's shares if event e_1 is happened before event e_2"*. Now let consider the Trading Engine Systems of two different firms A and B, both interested in e_1, e_2 and e_3. If one of the firms, say A, delivers events out-of-real time order, for example e_2 before e_1, then the event e_3 is not verified and the algorithm does not buy Microsoft's shares. This clearly leads to an unfair market between the firms A and B, that, on the contrary, delivered e_1 and e_2 in the correct real time order.

The data dissemination technology that can be used in stock market applications is DDS, due to the control of a rich set of QoS properties that makes it suitable for mission critical and real time systems. In particular, DDS can be used within Trading Engine Systems to allow communication among CEPD engines of the same firm, typically deployed in a local and well managed network. The communication over the Internet among Trading Engine Systems, Exchange and Real Time Market Data services also can use DDS; however, due to the unpredictable network behavior that may hamper the performance of the data dissemination technology, several Virtual Private Networks (VPNs) should be created in order to reduce the transmission delay and to have a higher control of the exchanged market data.

Algorithmic trading can exploit the high expressiveness communication model provided by DDS to define event subscriptions. Specifically, it can use a mix of channel- and content-based models: subscribers specify both their topics of interest and a range of values for the content of events published on those topics. Due to the high volume of information generated in the network, this two-level filtering ensures that each CEPD engine will be notified just about the subset of events that it will process. This requires no additional filtering on the subscribers' side, both reducing traffic and improving timely delivery.

Finally, it is worth noticing that DDS ensures only a per-source FIFO order. Thus, all entities in the electronic market must be synchronized with an external time source (i.e., GPS), and the ordering of events generated by different sources has to be managed at CEPD level, as currently provided by [Esper 2011].

5.3 Context-Aware Applications for Smart Environments

In the last few years, both the increasing availability of computational power and communication capabilities (wired and wireless) and the decreasing cost of small hardware devices, led embedded systems to gain popularity. Embedded systems are essentially made of specialized devices (e.g. sensors and actuators) used to control equipments such as automobiles, home appliances, communication, control and office machines, etc. Such devices are interconnected in order to enable their collaboration with the aim of providing complex functionalities.

These devices produce large amounts of data that can be leveraged by the system to sense the current status of the physical environment and react accordingly. This mass of information constitute the basis on top of which an application context can be built. The context represents the system's internal representation of the environment status and is used by the system to provide its service in a way that is correct, reliable and as efficient as possible. In particular, devices may have information about the circumstances under which they are able to operate and based on rules, or an intelligent stimulus, react accordingly enabling the creation of *context-aware* applications like smart-houses [SM4All 2008], smart-cities [SOFIA 2009], smart-hospital [Yao et al. 2011], just to cite a few.

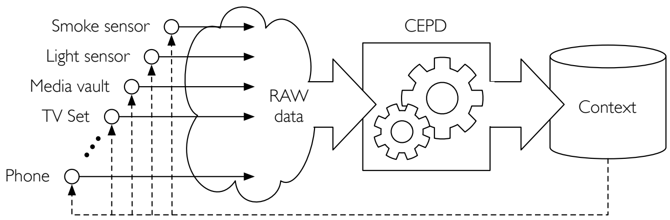

Figure. 4. Smart application scenario. Raw events produced by widely different sources are analyzed and processed to build a coherent context for the application.

As an example, let consider the following scenario: a person is at its automated home and decides to watch a movie. He would like to simply express this goal to the house (e.g., through a touch screen) and have the services of the house collaborate in order to accomplish this task. This high level goal is, in fact, constituted by many low level tasks such as: turn on the TV, show the list of available movies from the media server, reduce the light in the room by turning off the light or closing the curtains, divert all incoming phone calls to the voice mail service, etc. Some of these tasks should be executed in different ways depending on the current environment status: reducing the amount of light in the room can be accomplished by turning off the light only if the light is on because it is already night, otherwise closing the curtains is a better idea. An accurate and up-to-date context is thus necessary to fulfill the desired goal.

Another example, is represented by the smart-hospital application introduced in [Yao et al. 2011]. Hospitals are nowadays full of sensors of different types (e.g. physiological and environmental sensors) and more recently the Radio Frequency IDentification (RFID) technology started to be employed with the aim of identifying and tracking both people and objects. Consequently, a hospital needs to handle a large amount of data originated from a variety of sources, and needs to timely detect medically significant events by considering both RFID and non-RFID data. Let us consider, for example, the scenario where RFID readings can imply object movements and location changes: if an RFID observation indicates a wrong patient is taken into an operating room, a mismatch between the patient and the operating room can be detected triggering automatically and instantly an event to the medical staff warning it about the error.

A common characteristics of all the previous smart environment is represented by the need to correlate raw data and basic events to infer complex events or aggregated data before they can be used to create context information (e.g. it is day or night) or to detect potentially dangerous situation (e.g. a wrong patient has been moved to the operating room, or there is a fire spreading in the kitchen). A solution often adopted to collect data and build context information (see Figure 4) is based on the employment of a DS that routes raw data to a CEPD engine. The CEPD engine then analyses this data and produces context events that can be immediately consumed or stored for later use. Current CEPD engines are able to support the processing of event streams and the analysis of high-volume and high-speed data streams as required by the mentioned applications.

Required data dissemination properties. Considering the smart environments described so far, a DS collecting data from the embedded devices deployed in the environment should offer the following quality of service:

• *Best-Effort Delivery:* most information collected by sensors has a temporary validity and must be thus collected periodically (e.g. temperature). Sensors are configured to continuously push a stream of sensed data, thus the loss of few data items hardly affects the detection of complex events, but could rather delay it. This is a consequence of the fact that the periodic nature of this data induces an implicit retransmission mechanisms, that save the system from the need to implement any ad-hoc reliability solution. As an example, consider a scenario where every patient in an hospital is equipped with a RFID tag and each RFID reader sends information about identified tags once every 30 seconds. When a patient is moved from his room to the intensive care unit, the RFID tag will be first detected from the patient room reader and then later detected by the intensive care unit reader; however, after the movement, the patient will be detected every 30 seconds from the intensive care unit reader. As a consequence, in this case no reliable delivery guarantee is needed by the DS but rather it is sufficient to have best-effort guarantees due to the proactive read mechanism used by the RFID readers.

• *Real-Time Order*: ordering, and in particular real-time order, is strongly required by applications aimed at detecting *movement patterns*. As an example, consider again the scenario where every patient is equipped with a RFID tag. When a patient is moved from his room to the intensive care unit, the RFID tag will be first detected from the patient room reader and then from the intensive care unit reader. However, if the data dissemination service does not ensure real-time ordering, the two events can be delivered in a wrong order and the CEPD module could infer that the patient has been moved from the intensive care unit to his room, while he is supposed to remain there for urgent cares and a warning event can be triggered generating a false alarm.

• *Timely Delivery:* given the characteristic of context-aware applications to quickly react to the environment changes, most of the detection rules used by CEPD engines use temporal operators like the ones defined in Section 2 requiring, thus, a certain "degree of synchrony" among the events deliver times.

Currently, most of the smart environments described so far are built on top of ad-hoc sensors networks and the DS is implemented through specific multicast procedures.

5.4 Active Database in Cloud Computing

In the last few years, cloud computing emerged as a technology to provide resources on-demand and as a service over the Internet. Users can access these resources anytime and anywhere, both from desktops or mobile platforms. Amazon EC2, Google AppEngine, Microsoft's Azure are just a few examples of cloud architectures that provide services ranging from storage and application development to high speed computing platform. A public cloud is typically a complex infrastructure composed by one or more data centers, where a huge number of services runs on a large amount of hardware. A meaningful example of complex infrastructure is represented by the eBay architecture: in order to provide scalability, manageability and cost reduction, eBay made a functional segmentation of its enterprise into multiple disjoint subsystems: Users, Item, Transaction, Product, Account, Feedback (vertical division). Each subsystem is further divided into chunks (horizontal division) to parallelize the handling of requests within these [Shoup 2007].

The fundamental problem that arises from this segmentation, is the maintenance of data consistency among all the chunks in which a database is partitioned. Figure 5 depicts a cloud architecture where users, data management applications and the chunks themselves send events to consistently update all replicas of the same database.

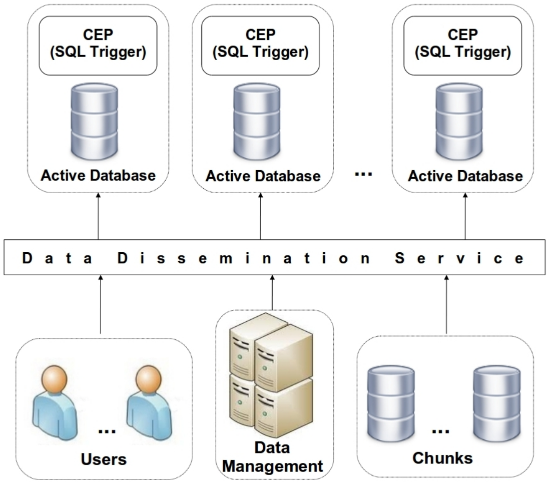

Figure. 5. Cloud architecture with several sources that send updates to a set of replicated active databases. CEPD modules execute security monitoring, alerting, statistics gathering and authorization.

Replicas (database chunks) can be viewed as active databases [Widom and Ceri 1996] that store all the same information. On top of them, CEPD modules implement SQL Triggers functionalities, i.e., persistent queries that are commonly used to audit changes, enforce and execute business rules, replicate data, enhance performance, monitor the application and gather statistics. Inconsistency could lead to detect some pattern in a chunk hosted in a data center, while leaving the pattern undetected in another data center. To prevent inconsistency, a typical approach is to use transactional ACID-based mechanism. However, this introduces an unsustainable load of interactions and synchronizations (e.g., locks) among cloud nodes that may also hamper the scalability of the system [Birman et al. 2009]. This is why major cloud providers are moving towards a decentralized convergence behavior in which cloud nodes are maintained in transiently divergent states, from which they will converge to a consistent state over time. This behavior is known as eventual consistency [Vogels 2009]: after an update completes, the system does not guarantee that subsequent accesses will return the updated value; there is an *inconsistency window* that represents the time period between an update and the moment in which any observer will always see the updated value [Vogels 2009].

A simple way to implement an eventual consistency algorithm is to use a best effort data dissemination service and rollback techniques that allow replicas to *correct* a wrong order. However, even if this solution avoids a strong coordination among nodes that would increase the risk that the whole cloud infrastructure may begin to thrash [Birman et al. 2009], it ensures that all replicas will be consistent only after a time t. Before t the result of persistent queries is unpredictable: in fact, the same pattern may be detected just by a subset of CEPD modules, due to the inconsistency of data stored in the databases.

To prevent this problem, the data dissemination service has to convey the high volume of events generated by sources in a reliable and totally ordered fashion. This two properties are fundamental for ensuring consistency without locking: reliability guarantees that all replicas will receive the same set of messages, while total order ensures that messages will be delivered in the same order by all receivers. This avoids the need for rollback techniques, because all chunks will see the same ordered sequence of updates, also preventing the unpredictability in persistency query results.

Reliable delivery and total ordering can be obtained at expenses of a timely delivery. In fact, as observed in [Birman et al. 2009], it does not matter how fast the protocol is; the aim is to avoid self-synchronization mechanisms among cloud nodes.

Required data dissemination properties. The data dissemination service used to support data consistency in cloud computing should guarantee the following properties:
- *reliable delivery:* each operation on a database triggers an update event that must be notified to all interest database chunks. The lack of reliability can have a strong impact on the final result of an operation. As an example, let us consider an auction bid with two bidders that issue an offer for the same product. The two bids represent events that must be notified to a set of database replicas. If one of the two events gets lost, then the outcome of the bid may not consider that offer, preventing the user from a possible victory.
- *total order:* in order to keep coherent copies of data in database chunks, events should be delivered in total ordering as defined in [Défago et al. 2004]. It does not matter that event deliveries follow the real time ordering of their emission, what matters is that every pair of events delivered by a pair of database chunks, are delivered in the same order.

Typically, the first generation of cloud computing used IP multicast as a means to disseminate information to every node in the system. However, the novel generation of clouds is banning this technology due to scalability issues with respect to the number of multicast groups [Birman 2009]: routers and Network Interface Cards (NICs) become promiscuous when a large number of multicast addresses is used, so as to swamp receivers with IP packets. This process leads receivers to drop packets, including good ones, that, in turn, causes massive spikes of NAK messages. Retransmissions just make things worse, due to the risk of multicast storm [Vigfusson et al. 2010]: a swamped receiver who has lost several packets will continuously require retransmissions to its multicast group, potentially provoking a storm of packets to all other nodes in the group, causing further drops. This process, obviously, generates throughput oscillations [Birman 2009].

For this reason, TCP is the favorite communication protocol for the new generation of cloud computing [Basin et al. 2010]. Even if the end-to-end delay of TCP links is higher than the transmission delay of multicast channels, stability matters more than speed [Birman 2009; Birman et al. 2009]. Moving from this consideration, a suitable solution for data dissemination in clouds is described in [Vigfusson et al. 2010], where processes use logical IP multicast addresses that are transparently mapped to real IP multicast addresses or one-to-one TCP connections. The use of IP multicast is limited to groups with similar interests, so as to merge them in a single group (non-interest messages are simply discarded at receiver's side). In addition, the diffusion algorithm imposes a control on the multicast rate, in order to prevent multicast storms and oscillations.

This mechanism, together with the use of TCP connections, ensures a reliable delivery, while ordering can be obtained by implementing a total order protocol on top of the data dissemination service.

6. CONCLUDING REMARKS

In the last years we are noticing an increasing use of Complex Event Processing applications for the monitoring of critical infrastructures deployed over large scale systems. CEPD sites correlate raw events incoming from different sources by means of several operators, and execute queries and rules in order to recognize complex patterns that could have spatial and/or temporal relationships.

A fundamental building block for CEPD-based applications is the data dissemination service used to convey raw events from sources to CEPD sites and the results generated by these sites to all intended destinations. This service has to provide several QoS requirements in order to sustain high-throughput systems and to allow the use of all operators at CEPD sites.

	Delivery policy		**Ordering policy**					**Timeliness**
DS Technology	**BE**	**Reliable**	**No Ord.**	**FIFO**	**Causal**	**Total**	**Real Time**	
DDS	✓	✓		✓				✓
JMS		✓		✓				
Multicast		✓		✓	✓	✓		
Gossiping	✓		✓					

Table I. Quality of Service policies satisfied by different data dissemination technologies.

	Delivery policy		**Ordering policy**					**Timeliness**
Case Studies	**BE**	**Reliable**	**No Ord.**	**FIFO**	**Causal**	**Total**	**Real Time**	
Collaborative sec.		✓		✓				second class
Stock market		✓					✓	first class
Smart env.	✓						✓	second class
Cloud Computing		✓				✓		

Table II. Quality of Service requirements for the analyzed case studies.

In this paper, we defined the main QoS requirements that the data dissemination has to satisfy, and grouped them in three distinct categories: *reliability, ordering* and *timeliness.* Then, we introduced the current data dissemination technologies that can be employed to design CEPD-based applications, and for each of them we illustrated the QoS requirements that they are able to satisfy. Finally, we presented four real case studies that use Complex Event Processing at their core, namely collaborative security, stock market, smart environments and active database, highlighting the requirements and the data dissemination technology suitable to address them.

The results of our study is summarized in tables I and II: the former evidences the QoS requirements addressed by the presented data dissemination technologies, while the latter shows the requirements of the four case studies. With reference to Table I, it is worth mentioning that *reliable delivery, FIFO, causal* and *total order* in the multicast technology can be obtained by implementing dedicated algorithms on top of the multicast protocol.

Looking closely at Table II, we notice that three out of the four case studies we have analyzed require the timeliness property. However, the electronic market is a *mission-critical* application, i.e., it imposes a very tight latency bound, typically of the order of milliseconds (referred to as *first class* in Table II). Collaborative security and smart environments applications, instead, impose a less stringent time bound, typically of the order of several seconds (referred to as *second class* in Table II). However, timeliness does not really matters for active database in cloud computing. In this scenario, timeliness is traded for scalability [Birman et al. 2009]: as such, consistency has to be ensured by avoiding the usage of locking algorithms that would impose an unsustainable load on the system due to the synchronization of cloud nodes. On the contrary, total ordering is of fundamental importance to guarantee consistency in all database replicas, preventing active queries from unpredictable results. Ordering is an important requirement also for collaborative security, stock market and smart environments. Stock market and smart environment applications need a real time order to detect temporal relationships among events occurred at different sources. The collaborative security scenario discussed in the paper, instead, does not correlate events coming from different source: CEPD sites are used to detect possible ongoing port scanning attack on single machines. As such, just a per-source FIFO ordering is required.

Differently from ordering, reliability matters just for the collaborative security, stock market and active database use cases, as a missing information could have a disruptive impact on the detection of an ongoing attack, on the stock trading and consistent updates. On the contrary, smart environments require just a best effort delivery; indeed, due to the periodic updates sent by sources, the loss of an event has no impact on the computation (a new sample simply overwrites the old value).

Starting from this analysis and focusing on QoS requirements satisfied by current data dissemination technologies (see Table I for reference), we can summarize that gossip is not suitable to implement a service that supports Complex Event Processing, due to its probabilistic nature. Despite smart environments do not require a reliable delivery, gossiping algorithms cannot be employed because they do not provide any form of ordering. DDS, JMS and multicast technologies are more suitable to address the requirements of CEPD-based applications. However, it is worth noticing that some functionality needs to be added to these technologies to fully meet all QoS requirements. Multicast protocols, as previously mentioned, do not provide native support for ordering and reliability: dedicated protocols can be employed to fill the gap. At the same time, DDS and JMS do not provide a real time ordering; thus, clock synchronization and event reordering have to be ensured at application level, as described in Section 5.2. Finally, JMS does not address timeliness requirements, so it cannot be used for mission-critical systems. A particular mention for the scalability of these technologies: some solutions such as DDS, JMS or IP multicast, provide good performance in a small and well managed environment, but they strongly degrade when deployed over a wide area network. This problem can be circumvented by using VPNs, as described in Section 5.2 for DDS, or by properly configure JMS servers. IP multicast, instead, cannot be deployed in WAN due to lack of network device support. As an alternative, application level multicast protocols are often employed. Several efforts have taking place to extend the deployment of data dissemination services from small to large scale systems still preserving the performance. An example is the BLEND European Project [PrismTech Ltd. 2010], whose aim is to design a discovery service and a data diffusion protocol for the applicability of DDS in large scale federated systems.

From the assessment of our study, we can conclude that none of the current data dissemination technologies can actually be used for CEPD-based applications without extending its functionalities at application level. As such, future research directions should concern the design of middleware services that provide a native support for QoS requirements, in particular timeliness and ordering, as highlighted by our analysis. In this way, QoS issues do not need to be addressed at application level, so as to leave CEPD sites to implement just their native functionalities (data correlation and complex pattern detection).

ACKNOWLEDGMENTS

This work has been partially supported by the Italian Ministry for Education, University, and Research (MIUR) in the framework of the Project of National Research Interest (PRIN) "DOTS-LCCI: Dependable Off-The-Shelf based middleware systems for Large-scale Complex Critical Infrastructures", by SM4All, Greener Buildings and BLEND Eurostar European Projects.

REFERENCES

Aniello, L., Luna, G. A. D., Lodi, G., and Baldoni, R. 2011. A collaborative event processing system for protection of critical infrastructures from cyber attacks. In *Proceedings of the 30th Conference on System Safety, Reliability and Security*. Napoli.

Balazinska, M., Balakrishnan, H., and Stonebraker, M. 2004. Contract-based load management in federated distributed systems. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation*. 197–210.

Baldoni, R., Beraldi, R., Tucci Piergiovanni, S., and Virgillito, A. 2005. On the modelling of publish/subscribe communication systems. *Concurrency and Computation: Practice and Experience 17,* 12, 1471–1495.

Baldoni, R., Contenti, M., Piergiovanni, S., and Virgillito, A. 2003. Modeling publish/subscribe communication systems: towards a formal approach. In *Proceedings of the 8th International Workshop on Object-Oriented Real-Time Dependable Systems*. IEEE, 304–311.

Baldoni, R., Querzoni, L., and Scipioni, S. 2008. Event-based data dissemination on inter-administrative domains: Is it viable? In *Proceedings of the 12th IEEE International Workshop on Future Trends of Distributed Computing Systems*. IEEE Computer Society, Washington, DC, USA, 44–50.

Basin, D., Birman, K., Keidar, I., and Vigfusson, Y. 2010. Sources of instability in data center multicast. In *Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware*. ACM, 32–37.

Birman, K. 2009. Rethinking multicast for massive-scale platforms. In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems.*

Birman, K., Chockler, G., and van Renesse, R. 2009. Toward a cloud computing research agenda. *SIGACT News 40,* 2, 68–80.

Birman, K. P. 1993. The process group approach to reliable distributed computing. *Commun. ACM 36,* 12, 36–53, 103.

Birman, K. P. and Joseph, T. A. 1987. Exploiting virtual synchrony in distributed systems. In *Proceedings of the 11th ACM Symposium on Operating System Principles.* 123–138.

CoMiFin. 2008. CoMiFin - Communication Middleware for Monitoring Financial Critical Infrastrucures. http://www.comifin.eu/.

Corsaro, A., Querzoni, L., Scipioni, S., Piergiovanni, S. T., and Virgillito, A. 2006. Quality of service in publish/subscribe middleware. In *Global Data Management*, R. Baldoni and G. Cortese, Eds. IOS Press.

Costa, P. and Picco, G. P. 2005. Semi-probabilistic content-based publish-subscribe. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems.* IEEE Computer Society, Washington, DC, USA, 575–585.

Défago, X., Schiper, A., and Urbán, P. 2004. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys (CSUR) 36,* 4, 372–421.

Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. 1987. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th annual ACM Symposium on Principles of distributed computing.* ACM, 1–12.

Deutzman, J. 2010. FBI investigates $9 Million ATM scam. http://www.myfoxny.com/dpp/news/090202_FBI_Investigates_9_Million_ATM_Scam.

Esper. 2009. Where Complex Event Processing meets Open Source: Esper and NEsper. http://esper.codehaus.org/.

Esper. 2011. Esper time-order. http://esper.codehaus.org/esper-4.2.0/doc/reference/en/html_single/index.html#view-time-order.

Etzion, O. 2008. Event processing architecture and patterns. In *Proceedings of the 2nd International Conference on Distributed Event Based Systems.*

Huang, Y., Feamser, N., Lakhina, A., and Xu, J. J. 2007. Diagnosing network disruptions with network-wide analysis. In *SIGMETRICS'07.* San Diego, California, USA.

JBoss. 2010. JBoss Drools Fusion. http://www.jboss.org/drools/drools-fusion.html.

JGroups. 2010. Jgroups - a toolkit for reliable multicast communication. http://www.jgroups.org//.

Liebig, C., Cilia, M., and Buchmann, A. 1999. Event composition in time-dependent distributed systems. In *Proceedings of the 4th IECIS International Conference on Cooperative Information Systems.* IEEE Computer Society, Washington, DC, USA, 70.

Lodi, G., Aniello, L., and Baldoni, R. 2011. Inter-domain stealthy port scan detection through complex event processing. In *Proceedings of the 13th European Workshop on Dependable Computing.* ACM, Pisa, 67–72.

Lodi, G., Querzoni, L., Baldoni, R., Marchetti, M., Colajanni, M., Bortnikov, V., Chockler, G., Dekel, E., Laventman, G., and Roytman, A. 2009. Defending financial infrastructures through early warning systems: the intelligence cloud approach. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies.* ACM, 18.

Microsystem, S. 2008. Java Message Service (JMS). http://java.sun.com/products/jms/.

NESSI. 2009. NESSI Strategic Agenda.

Palmer, M. and Dzmuran, M. 2009. An Introduction to Event Processing, white paper. http://www.cnetdirectintl.com/direct/fr/2009/progress/0907_centre_ressources/ressources/1-1_LB_UK_introduction_CEP.pdf.

Pietzuch, P., Shand, B., and Bacon, J. 2004. Composite event detection as a generic middleware extension. *Network, IEEE 18,* 1, 44–55.

Pietzuch, P. R. 2004. Hermes: A Scalable Event-Based Middleware. Ph.D. thesis, University of Cambridge.

PrismTech Ltd. 2010. BLEND Box. http://www.prismtech.com/news/.

Rao, S. K. 2010. Algorithmic trading: Pros and cons. http://www.tcs.com/SiteCollectionDocuments/WhitePapers/TCS_FS_algorithmictrading.pdf.

Shoup, R. 2007. Randy Shoup on eBay's Architectural Principles. http://www.infoq.com/presentations/shoup-ebay-architectural-principles.

SM4All. 2008. SM4All Project - Smart Homes For All. http://www.sm4all-project.eu/.

SOFIA. 2009. SOFIA Project - Smart Objects For Intelligent Applications. http://www.sofia-project.eu/.

Van Renesse, R., Birman, K. P., and Vogels, W. 2003. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst. 21,* 2, 164–206.

Vigfusson, Y., Abu-Libdeh, H., Balakrishnan, M., Birman, K., Burgess, R., Chockler, G., Li, H., and Tock, Y. 2010. Dr. multicast: Rx for data center communication scalability. In *Proceedings of the 5th European conference on Computer systems*. ACM, 349–362.

Vijayan, J. 2004. Update: Credit card firm hit by DDoS attack. `http://www.computerworld.com/securitytopics/security/story/0,10801,96099,00.html`.

Vogels, W. 2009. Eventually consistent. *Commun. ACM 52,* 1, 40–44.

Widom, J. and Ceri, S. 1996. *Active database systems: Triggers and rules for advanced database processing.* Morgan Kaufmann Pub.

Yao, W., Chu, C.-H., and Li, Z. 2011. Leveraging complex event processing for smart hospitals using RFID. *J. Netw. Comput. Appl. 34*, 799–810.

Zhang, G. and Parashar, M. 2010. Cooperative detection and protection against network attacks using decentralized information sharing. *Cluster Computing 13,* 1 (March), 67–86.

Roberto Baldoni is a Full Professor in Computer Science at Sapienza - Universita' di Roma. He conducts research (from theory to practice) in the fields of distributed, pervasive and p2p computing, middleware platforms and information systems infrastructure with a specific emphasis on dependability and security aspects. Roberto Baldoni is a member of the IFIP WG 10.4, of the steering committees of ACM DEBS, DSN and of the editorial boad of IEEE TPDS.

Silvia Bonomi is a PhD in Computer Science at the Sapienza - Universita' di Roma. She is doing research on various computer-science fields including dynamic distributed systems and event-based systems. In these research fields, she published several papers in peer reviewed scientific forums. As a part of the MIDLAB research group, she is currently involved in an EU-funded project dealing with energy saving in private and public buildings(GreenerBuildings project) and she worked on dependable distributed systems (ReSIST network of excellence) and on the definition of new semantic tools for e-Government (SemanticGov).

Giorgia Lodi is currently a consultant at DigitPA, the Italian government agency responsible for the digitalization process of Public Administration. She received the Ph.D. degree in Computer Science from the University of Bologna (Italy). From 2001 to 2002 she was a research associate of Computer Science at the University of Newcastle upon Tyne (U.K.). From September 2008 to June 2011 she was a Post-doc researcher at Sapienza - Universita' di Roma. In the recent past she published peer-reviewed papers in various computer-science fields. Her principal research interests include cloud computing, e-government services, security, complex event processing.

Marco Platania is a PhD student at Sapienza - Universita' di Roma. He obtained the Laurea degree in computer engineering from the same institution in 2008 with a work on internal clock synchronization in presence of byzantine faults in WAN environment. His research activity covers peer to peer systems, middleware and event-based systems. As member of the MIDLAB research group, he is involved in Italian and European projects about diagnosis and monitoring of critical infrastructures, peer to peer solutions for next generation networks and Data Distribution Service in ultra large scale systems.

Leonardo Querzoni is an Assistant Professor in Computer Science at Sapienza - Universita' di Roma. He obtained a PhD in in computer engineering from the same institution with a work on efficient data dissemination though the publish/subscribe communication paradigm. His research activities are focussed on several computer science fields including large scale and dynamic distributed systems, dependability in distributed systems, publish/subscribe middleware services, wireless and sensor networks. He regularly serves in the technical program committees of several conferences in the field of dependability, event-based communications and autonomic systems, and collaborates as a reviewer for international conferences and journals of his research areas. Being part of the MIDLAB research group, he is currently involved in several international EU-funded projects dealing with dependability and performance in smart-spaces and home automation (SM4All, eDIANA and SOFIA projects).