

Chronicle Recognition for Mobility Management Triggers

Christophe Dousson
France Telecom R&D
2, avenue Pierre Marzin
F-22307 Lannion Cedex, France
christophe.dousson@orange-ftgroup.com

Kostas Pentikousis, Tiia Sutinen, and Jukka Mäkelä
VTT Technical Research Centre of Finland
Kaitoväylä 1
FI-90571 Oulu, Finland
firstname.lastname@vtt.fi

Abstract

An optimally working mobile system requires tight cooperation and an information stream that flows impeccably between its components. This, however, is not the current state of the art, as applications and system components must each, in isolation, acquire data, create a knowledge base, and maintain it with diligence. We introduce chronicle recognition to our trigger management framework and present the advantages of generating synthetic triggers based on events and measurements, establishing the theoretical foundation. We explain how, by using a time point algebra, we can reason with reified logic and obtain triggers that are unavailable otherwise. These triggers can be generated efficiently, as our results indicate, and are reliable in terms of event temporal correlation.

1. Introduction

Mobile hosts face an environment with several technical limitations and high user demands. Users ask for continuous, affordable, and seamless connectivity. Mobile platforms often have to deal with spotty network coverage and the variability of wireless channel conditions, address security concerns, cope with battery power consumption, and choose the best-connected network attachment, while running a variety of applications, including instant messaging and VoIP clients and, increasingly, mobile peer-to-peer applications. State of the art applications with stringent requirements, such as VoIP, typically measure end-to-end (e2e) connection characteristics (say, delay and packet losses) in order to adjust sending rates and choose the most appropriate codec. Meanwhile, network interfaces measure access link variants, such as, received signal strength, frame retransmissions, and other MAC-related metrics. UMTS interfaces can also acquire information about the cell load over the air. If the wireless signal strength deteriorates during a VoIP call, the client observes degraded e2e perfor-

mance, but it is not alerted about first-hop failings immediately. It needs to determine the currently used network interface, poll it, and correlate its own measurements with those of the interface. If the first hop is clearly underperforming, there is little gain from taking any action to address the measured e2e degradation. A vertical handover may be much more effective, for example. On the other hand, if the link is working properly, the VoIP client can take proper e2e actions, say, by adjusting its sending rate and mode.

Other applications and system components, such as mobility management protocols, may also need to act similarly and acquire information from different components in order to make optimal handover (HO) decisions. We argue that HO initiation should be controlled by intelligent decision-making based on user, application, and operator policies and preferences, network resource availability, and user device capabilities rather than blindly roaming to a new network when it becomes available. In previous work [1,2] we introduced the mobility trigger management mechanisms which provide the basis for this kind of sophisticated mobility management. This paper builds upon these and introduces chronicle recognition (CR) for mobility management.

2. Background and Motivation

Both research and standardization work (see, for example, [2–6] and the references therein) concur that mobile devices require more information to work optimally. For instance, the IEEE 802.21 working group is looking into ways to enable HO and interoperability between heterogeneous network types, including what information ought to be collected and disseminated; however, *how* is this information collected, disseminated, and used is not addressed. By combining the generic framework for mobility-related trigger collection, temporary storage, and delivery [1] with a context management system, we defined a service capable of maintaining and delivering up-to-date information to different system and network entities [2]. The term *trigger* in this case refers to a notification generated when system

changes occur. Information related to the current system state is referred to as context. This framework leaves the final interpretation of the trigger information and the associated decision-making with the trigger consumer, mainly considering the case of a consumer receiving triggers from several sources at some point in time. A mobility management protocol, for example, will receive a trigger whenever a change relevant for its operation occurs, say, when a link goes up/down or the respective signal strength drops from “high” to “low”. Temporal correlation of individual triggers is not supported. Often, however, consumers may want to receive a single trigger, indicating that a temporally correlated set of triggers has been detected, instead of receiving a multitude of them and inferring the end result. We refer to triggers generated based on two or more other temporally correlated triggers as *synthetic triggers* in the remainder.

A typical use case for synthetic triggers is when the current link is still usable and the HO decision-making logic requires more than one condition to be met before it will make a positive HO decision (say, the reception of a WLAN link up trigger will cause a HO from the currently used UMTS link to WLAN iff the latter link quality is “high” and the user policy allows it). On the other hand, imperative HOs (i.e. the current link has become unusable) are initiated after receiving a single “link down” or radio link quality trigger. Moreover, synthetic triggers could be used to prevent unnecessary HOs by detecting unstable links (e.g. a number of consecutive “link up/down” triggers received in a set time window can be interpreted as an “unstable link”); or, the “ping-pong effect” (i.e. a number of consecutive vertical HOs between two access points within a short period).

Note that the reception of a link layer (L2) trigger does not necessarily indicate that the link is usable by the mobile host. Mobility management can benefit from information about the existence of firewalls and other access limitations used in the network that may hinder its operation. One may need to wait for other triggers possibly following the L2 one (such as, routing table updates, security association establishment, availability or lack of VPN connectivity, and warning(s) about limited Internet access due to firewall restrictions) before making a HO decision. Although the trigger consumers of [1, 2] have access to a wide variety of cross-layer and cross-domain information, they may still need to maintain their own repositories of received triggers and introduce logic to combine the information extracted from them. By adding chronicle recognition (§3) to our trigger management framework we can generate synthetic triggers, which can address these issues as explained in §4.

3. Chronicle Representation and Recognition

As we want to detect particular evolutions of a system, the critical instants are those corresponding to (partial) changes of its state. Using a formalism such as the one

presented in [7, 8] we can track the system evolution and model changes via correlation rules called *chronicles*. The chronicle formalism was inspired by the temporal reified logic proposed in [9], although due to the time constraint management complexity, temporal information relies on a time point algebra [10] instead of time intervals. For instance, the received signal strength indication can be represented by attribute S . A definition domain \mathcal{D} is associated to each attribute; for S we have $\mathcal{D}_S = \{low, average, high\}$. Similarly, the battery state of charge (SOC) attribute B has domain $\mathcal{D}_B = \{critical, low, average, high, charging\}$, while the current link attribute L has domain $\mathcal{D}_L = \{N/A, bad, average, good\}$.

A chronicle is defined as a set of trigger patterns which are temporally constrained and related to a change of value (from v_1 to v_2) of an attribute A at time t , noted in reified logic with the associated predicate $event(A : (v_1, v_2), t)$. Persistence of the value of an attribute between two instants is expressed with the assertion predicate: $hold(A : v, (t_1, t_2))$, while absence of a trigger between two instants is denoted with $noevent(A : (v_1, v_2), (t_1, t_2))$. For example, consider a scenario involving three triggers. First, a trigger informs us that the battery SOC is critical; another one (received within 2 to 5 time units later) indicates that connectivity becomes shaky (“good” to “bad”) and, less than 10 time units since the first trigger, we detect high IP packet loss. This chronicle is defined as follows:

```
chronicle myChronicle {
  event(B:(?, critical), t1)
  event(L:(good, bad), t2)
  event(IPloss:(?, high), t3)
  hold(link:WiFi, (t1, t4))
  t2 - t1 in [2,5]
  t2 < t4
  t3 - t1 in [0,10]
  t3 < t4
}
```

Given an input stream of triggers and a chronicle we can extract [8] all subsets matching it. For illustrative purposes, consider the chronicle to detect the start sequence of, at least, three consecutive “link up/down” triggers within 30s, which is good indication of an unstable link. First, we define a chronicle which detects a single link up/down (LUD) emitting a new trigger on each recognition and then we recognize two sequences only if the time between them is at least 60s:

```
chronicle LUD(t1) {
  event(link:(down, up), t1)
  event(link:(up, down), t2)
  hold(link:good, (t1, t2-1))
  0 < t2 - t1 < 5 -- up/down within 5s
}
chronicle LinkUnstable {
  event(LUD,t1) noevent(LUD, (t1-60,t1-1))
  event(LUD,t2) noevent(LUD, (t1+1,t2-1))
  event(LUD,t3) noevent(LUD, (t2+1,t3-1))
  t1 < t2 < t3
  t3 - t1 < 30
}
```

Fig. 1 illustrates this example recognition using filled circles (●) for the *LUD* triggers and stars (★) for the recognized chronicle “LinkUnstable”. Note that at recognition time, synthetic triggers corresponding to both types of chronicles will be generated, and disseminated to consumer(s) based on their subscriptions [1, 2]. In general, triggers may be shared by many chronicles and the system ought to be able to manage all concurrent instances. Note that recognition is exhaustive, that is, all possible instances of the defined chronicles are identified by the system (see Fig. 1). The recognition process manages a set of partial instances of chronicles as a set of time windows (one for each forthcoming trigger), which is gradually constrained by each new matched trigger. This is predictive in the sense that it anticipates forthcoming triggers relevant to instances of chronicles currently under consideration. CR focuses on these, maintains their corresponding time windows and is able to deal with “jittered” input as incoming triggers are processed on the fly even if they arrive in a non-chronological order it (for more details see [7, 8]). We will see in the following section that these properties are necessary for recursive correlation.



Figure 1. Chronicle recognition illustrated

4. Mobility Triggers

To perform correlation of triggers (i.e. CR), we should define what the input triggers are and how are they produced. At the lowest level, we have producers which emit “raw” data or “basic” triggers. These triggers are collected, processed and disseminated to consumers by a functional entity called TRG. In this schema, the CR component of TRG acts as an internal consumer of “basic” triggers depending on the defined chronicles and acts as producer of new, synthetic triggers, issued once a CR occurs. In the example of §3, CR consumes “link up” and “link down” triggers, and produces two types of synthetic triggers (*LUD* and *LinkUnstable*) once the corresponding chronicle is recognized. Note that CR stores information related only to triggers considered by at least one chronicle and recall that the system reasons only on changes. We first process incoming triggers to detect significant changes: a discretization step is performed in order to deal with an adequate range of values and then compute when the measurement changes against these values. For instance, if the radio signal quality is measured from 0 to 100, we define three ranges: [0, 40) (low), [40, 80) (average), and [80, 100] (high). When the radio signal goes through a threshold, a trigger is generated. For the signal depicted in Fig. 2 the

evolution is summed up with two triggers: $event(signal : (low, high), 110)$ and $event(signal : (high, low), 117)$. These triggers alone should be considered in CR, although TRG consumers can also opt to receive all measurements.

The CR implementation receives chronicles from different consumers and includes them all in its reasoning. We highlight that this provides flexibility and allows different consumers to create their own synthetic triggers: from one run to another, each consumer chooses what is important to monitor and provides a chronicle for it. This keeps memory requirements to the bare minimum, which is important for small devices. This also delegates a part of the reasoning on the observations: the consumer does not need to manage a buffer of triggers nor track what is happening; it only needs to define its own chronicles and wait for the synthetic trigger. Note that the number of *attributes* in the system depends on the producers (which generate basic triggers), but not on the number of chronicles defined by the consumers.

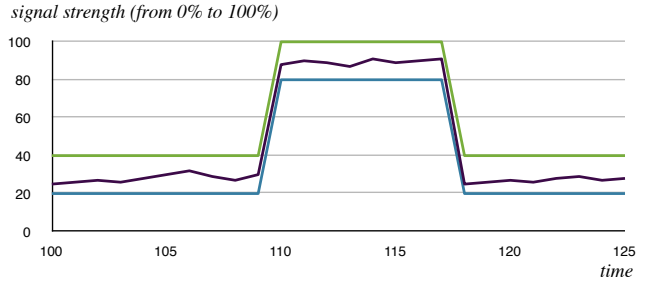


Figure 2. Raw data and discretization

4.1. Synthetic Trigger Generation

The main reason for using CR is that it is more than adequate to, first, temporally correlate possibly disparate triggers originating from different parts of the protocol stack and other domains and, then, generate a synthetic trigger, which can be subsequently used to produce other synthetic triggers using second or higher order chronicles. As all triggers are timestamped, the recognition timestamp should be related to one specific trigger in the chronicle: this ensures that the definition of the chronicle timestamp is absolute and does not depend on when the CR computational process actually completes. For example, consider a synthetic trigger generated when two triggers a and b occur within 5 to 10 time units apart; the corresponding chronicle ξ is

$$\xi : event(a, t) \wedge event(b, t') \wedge (5 \leq t' - t \leq 10)$$

After a and b are received, ξ will be recognized and a synthetic trigger is generated. However, the current system time may differ from t or t' (say, due to the delay in transmission/processing of a and b). If we use the current time

to timestamp the synthetic trigger, we will generate different timestamps for the same (a, b) , depending of how much time we spent computing the correlation (or waited for triggers). This is not reasonable: if this trigger will be used in another correlation, we will end up with inconsistencies in correlating correctly synthetic triggers because their timestamps are unreliable. For instance, consider the chronicle χ , which is based on triggers generated by ξ :

$$\chi : event(\xi, t) \wedge event(b, t)$$

If ξ is timestamped when b is received, then this rule can be applied. However, if there is some delay between the occurrence of b and the generation of ξ , then it cannot apply even if the actual system behavior is the same. In this case, the observed difference is only due to the implementation of the trigger collection and correlation mechanisms, making it impossible to define a set of consistent chronicles. That is, the timestamp of each synthetic trigger should be specified in a manner relative to the triggers involved in each correlation. Thus, we set the timestamp in relation to an actual trigger and not on current time, shielding CR from computational delays. For example, if the synthetic trigger ξ were to be associated with b , we have:

$$\xi(t') : event(a, t) \wedge event(b, t') \wedge (5 \leq t' - t \leq 10)$$

Similarly, we can choose a as the “major” trigger in the chronicle and generate $\xi(t)$ with the time of a . We can also add a fixed delay between the timestamps: for instance, we could generate $\xi(t'')$ with $t'' = t + c$, with c a constant. In all these cases, the timestamp is properly defined. Note that, as a consequence, the reception and occurrence times of a synthetic trigger may differ and so, in general, the generation of a trigger does not necessarily follow real-time order. This also means that we ought to use a CR implementation which does not assert chronologically ordered triggers.

Finally note that the expressiveness of the formulation presented in §3 allows us define a synthetic trigger which includes both occurrences of triggers and also non-occurrences of certain (other) triggers. For example, we define $\xi(t)$ s.t. a is observed at t and is not followed by b

$$\xi(t) : event(a, t) \wedge noevent(b, (t, t + 5))$$

In this case, a trigger will be generated only when we are sure that no b triggers occur, and its timestamp will be the same as the timestamp of a . Once again, the actual time of the synthetic trigger generation is not relevant.

4.2. Recursive Synthetic Triggers

As mentioned above, synthetic triggers can be consumed by other chronicles (as any other trigger). If a synthetic trigger is generated by a chronicle defined using solely “basic”

triggers, it is sufficient to generate it when a recognition is completed and reinject it back into the CR implementation. However, if the generating chronicle is based other synthetic trigger(s), correct recognition becomes complicated. For instance, if a chronicle employs the *hold* or *noevent* statements, a CR implementation following solely what was described above can run into a deadlock. To elucidate this, let us define two exclusive chronicles by using the *noevent* statement as a guard condition:

$$\begin{aligned} \chi(t) & : \dots \wedge noevent(\xi, (t, t')) \wedge (0 \leq t' - t \leq 10) \\ \xi(u) & : \dots \wedge noevent(\chi, (u, u')) \wedge (0 \leq u' - u \leq 10) \end{aligned}$$

In this case, ξ and χ cannot conclude even when all other triggers are received, because the chronicles use what we call *recursive* synthetic triggers. To avoid this deadlock, we maintain forthcoming time windows for each chronicle time point as explained in [11], that is, for each pending (or partial) instance χ_i of chronicle $\chi_i(t)$, we compute the tightest time window for $TW(\chi_i, t)$. As a result, all times that χ can be produced are included in:

$$TW(\chi) = \bigcup_{\chi_i} TW(\chi_i, t)$$

In this case, χ cannot be generated at a time in the complementary \mathcal{C} of $TW(\chi)$ so the instruction *assertNoMoreEvent* $(\chi, \mathcal{C}(TW(\chi)))$ can be sent to CR which uses it to avoid unnecessary pending instances as described in [11]. Of course, a similar process should be performed also for ξ and for each synthetic trigger. We can follow this approach to dispose of other useless pending chronicles: if the synthetic triggers are involved in only one other chronicle (i.e. are not used by an external consumer), we can use the time window information to compute when they should be discarded. In the previous example, all instants of χ which could affect the *noevent* predicate in the chronicle ξ are included in the following forthcoming window:

$$FW(\xi, \chi) = \bigcup_{\xi_i} [min(TW(\xi_i, u)), max(TW(\xi_i, u'))]$$

Consequently, all partial instances of χ_i generating χ outside this window can be deleted even if they lead to a recognition. The best way to do this is to reduce the time window of t for all instances (and the CR implementation will propagate this reduction to other chronicle instants):

$$\forall i, TW(\chi_i, t) \leftarrow TW(\chi_i, t) \cap FW(\xi, \chi)$$

These two proposed extensions are sufficient to solve the deadlock problem of recursive synthetic triggers. Note that these extensions do not generate more pending chronicles but only “assertNoMore” instructions which lead to deletion of pending instances (see [11] for more details) and, thus, the use of recursive triggers does not affect CR performance significantly.

5. Results

This section reports on results from the first phase of integration between TRG [1] and the CRS implementation of CR [8], performed as part of the mobility management work in the Ambient Networks project [6]. Our integration work is ongoing and, at this stage, we can only report on simulated tests, aiming at affirming the feasibility of our approach. Previous experiments [11] showed that CRS can process an input rate of hundreds of triggers per second (which is sufficient for our mobility management mechanisms), but did not focus on the memory consumption aspect, which is important in our context since we have deal with small devices as well.

The results presented below are produced by having CRS recognize sequences of triggers from a log containing 1 000 000 triggers of 8 types (from a to h), which is uniformly randomized. The chronicles to match are sequences (for instance $a \rightarrow b \rightarrow c$) with $[0, 2\delta]$ as a time constraint between two successive triggers where δ is the average distance between two successive triggers in the log. Examining more realistic trigger logs are part of our future work.

First we point out that processing time for this log is linear with the number of chronicle patterns. More precisely, if $\tau(\xi_i)$ is the processing time for chronicle ξ_i (for sequences of three triggers, it takes about 10s per chronicle to process 1 000 000 triggers on JVM 5.0 on MacOSX, G5 2GHz.), then the total processing time of n chronicles is $\sum_{i=0}^n \tau(\xi_i)$. We observe the same linear property for memory size. Second, but more interesting to measure is how CRS behaves when the recognition sequence size grows. CRS maintains all required information about partial instances of chronicles (the *pending chronicles*), so counting them gives a reliable estimate about memory consumption. To estimate used memory, we record, after each log trigger, how many pending instances are still under consideration; thus we collect 1 000 000 samples.

Table 1 presents results when the recognized sequences contain 2-8 triggers, while Table 2 presents our memory evolution results when the constraints between two successive triggers of the same sequence pattern grows (from $[0, \delta]$ to $[0, 10\delta]$). In short, performance is good when the number of triggers of a sequence grows: both the mean and the standard deviation are quite low. For a given number of triggers, increasing the time constraint has no significant impact on performance which grows linearly even if the number of recognized chronicles increases faster.

In short, average memory consumption remains manageable even if the patterns grow in size and, as previous experiments show, the implemented recognition process supports high input rates (hundreds of triggers per second), thus allowing for real-time processing. Nevertheless, these tests are still preliminary and we are currently proceeding with testing CRS with chronicle and trigger logs originating from

Table 1. Memory consumption as a function of the current number of pending chronicles

Seq. length	Max duration of chronicle	Process time	Pending chronicles	
			<i>mean</i>	<i>std dev</i>
2	20 s.	10 s.	1.36	0.72
3	40 s.	12 s.	1.56	0.53
4	60 s.	24 s.	1.73	0.29
5	80 s.	33 s.	1.89	0.73
6	100 s.	40 s.	2.09	1.04
7	120 s.	56 s.	2.30	1.34
8	130 s.	66 s.	2.55	1.68

Table 2. CRS memory consumption is linear

Constraint length	Recognitions count	Pending chronicles	
		<i>mean</i>	<i>std dev</i>
$\delta = 10$	18	1.74	0.61
$2.\delta = 20$	249	1.32	0.85
$3.\delta = 30$	991	1.47	1.09
$4.\delta = 40$	2 414	1.64	1.32
$5.\delta = 50$	4 804	1.82	1.55
$6.\delta = 60$	7 860	2.00	1.78
$7.\delta = 70$	11 679	2.18	1.99
$8.\delta = 80$	16 138	2.36	2.21
$9.\delta = 90$	20 727	2.54	2.40
$10.\delta = 100$	25 813	2.71	2.60

the mobility management context presented earlier.

6. Related Work

Although the core of our system resembles a standard notification system with similar operating principles as in the systems presented in [12–14], certain aspects render it unique. First, it is designed for dissemination of event information in an extremely diverse environment consisting of heterogeneous networks and terminals. Second, our system specializes in mobility related triggers and complements well, for example, the upcoming IEEE 802.21 standard by enabling event signaling throughout the protocol stack from the physical and link layers up to the applications. CRS, on the other hand, is a temporal event correlation system; its novelty is that it uses time information to focus on particular time windows and then reasons on the atemporal part. All previously implemented correlation systems do the opposite: [15] presents a panel of solutions for event correlation in a telecommunications context but none of them relies on time information. They work similarly to rule-based or case-base reasoning systems. Some of them extend their formalism to represent time (e.g. [16] for ILOG/Rules) but

the time constraints are always checked after receiving all relevant triggers, they are not used to alleviate the recognition process, and can not be used to remove indecision in some cases as shown in Section 4.2.

7. Conclusion and Future Work

We introduced how a trigger management architecture can take advantage of CR in generating synthetic triggers based on other triggers and measurements, establishing the theoretical framework foundation. We showed that by using a time point algebra we can reason with reified logic and obtain triggers that are both reliable (in terms of event temporal correlation) and unavailable otherwise. However, although initial implementations, prototypes used in a testbed, and simulation studies (partially presented in this paper) are quite promising, we still have some way to go until we have a complete product, ready to be integrated in mobile operating system kernels, and great challenges lie ahead. First, we aim at serving extremely demanding system components that deal with mobility management. This implies that as we move forward, implementation details need to be carefully taken care of. Furthermore, CR has not been customized yet for small mobile devices; this is part of our ongoing effort. We want to make sure that synthetic triggers can be supported even on small footprint devices. Optimizations in message exchanges and method calls, to name a few, are the forefront of future work. Although studying scalability issues are high on our agenda as well, we are also looking intensively into implementing more producers and consumers.

Acknowledgement

We thank the anonymous reviewers for their comments and suggestions; they have made this paper significantly better. This work has been carried out in the framework of the Ambient Networks project, which is partially funded by the Commission of the European Union. The views expressed in this paper are solely those of the authors and do not necessarily represent the views of their employers, the Ambient Networks project, or the Commission of the European Union.

References

- [1] J. Mäkelä and K. Pentikousis. Trigger management mechanisms. In *Proc. International Symposium on Wireless Pervasive Computing*, Puerto Rico, 2007.
- [2] R. Giaffreda, K. Pentikousis, E. Hepworth, et al. An information service infrastructure for Ambient Networks. In *Proc. IASTED Parallel and Distributed Computing and Networks*, Innsbruck, Austria, Feb. 2007.
- [3] JH. Choi and G. Daley. Goals of detecting network attachment in IPv6. IETF RFC 4135, Aug. 2005.
- [4] S. Schütz, L. Eggert, S. Schmid, and M. Brunner. Protocol enhancements for intermittently connected hosts. *Comput. Commun. Rev.*, 35(3):5–18, 2005.
- [5] V. Typpö, O. Gonsa, T. Rinta-aho et al. Triggering multi-dimensional mobility in Ambient Networks. In *Proc. 14th IST Mobile & Wireless Communications Summit*, Dresden, Germany, June 2005.
- [6] N. Niebert et al. Ambient Networks – An Architecture for Communication Networks Beyond 3G. *IEEE Wireless Communications*, 11(2):14–21, 2004.
- [7] C. Dousson. Alarm driven supervision for telecommunication networks : II- On-line chronicle recognition. *Annals of Telecom.*, 51(9/10):501–508, 1996.
- [8] CRS. A Chronicle Recognition System. *The Official Web Site*. <http://crs.elibel.tm.fr>.
- [9] Y. Shoham. Temporal logics in AI: semantical and ontological considerations. *Journal of artificial intelligence*, 33:89–104, 1987.
- [10] D. V. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.
- [11] C. Dousson and P. Le Maigat. Improvement of chronicle-based monitoring using temporal focalization and hierarchization. In *Proc. 17th International Workshop on Principles of Diagnosis*, June 2006.
- [12] G. Cugola, E. Di Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE Transactions on Software Engineering*, 27(9):827–850, 2001.
- [13] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proc. Symp. on Principles of Distributed Computing*, 2000.
- [14] R. Meier and V. Cahill. Steam: Event-based middleware for wireless ad hoc networks, 2002.
- [15] G. Jakobson. The technology and practice of integrated multiagent event correlation systems. *Integration of Knowledge Intensive Multi-Agent Systems (KIMAS)*, pp. 568–573, October 2003.
- [16] B. Berstel. Extending the RETE algorithm for event management. *9th International Symposium on Temporal Representation and Reasoning (TIME)*, 2002.