A reconfigurable distributed CEP middleware for diverse mobility scenarios

Piotr Kamisiński, Vera Goebel and Thomas Plagemann
Department of Informatics, University of Oslo, Norway
{*piotr, goebel, plageman*}*@ifi.uio.no*

***Abstract*—Sensor nodes and complex event processing (CEP) are important and powerful means for gathering data and detecting phenomena of interest in mission-critical pervasive systems, e.g. for emergency and rescue operations. However, the dynamic network does not allow using centralized CEP. To address this issue, we present a component-based distributed CEP middleware. Its main goal is easy reconfigurability to different mobility scenarios. This is achieved by providing an extensible collection of algorithms that are tailored for specific scenarios. The middleware makes it possible to select on demand the algorithms that are most suitable for the current scenario. Our evaluation shows that the middleware works in a broad spectrum of mobility scenarios. We also investigate the trade-off between efficiency and reliability of distributed CEP.**

I. Introduction

Complex event processing (CEP) is a powerful tool to detect events of interest based on data from multiple data sources, e.g. sensors. The user specifies events in form of subscriptions. Given a set of subscriptions, the CEP system processes data. When an event is detected, a notification is issued and presented to the user. CEP is used in many application domains, e.g. business process monitoring, or automated home care [7]. The CEP solutions used there are typically centralized, i.e. all data from sources is sent to and processed by a central CEP system.

In some new application domains for pervasive computing, e.g. emergency and rescue (E&R) or environmental monitoring, centralized CEP performs badly or does not work at all. The main characteristics that contribute to this are: possible network partitions, failures of computing nodes, and diversity of scenarios (mobility, resources, workloads).

In E&R operations, it is not possible to rely on a network infrastructure because it might not exist in remote areas, or might be damaged or congested. Instead, data from sensors can be collected via a mobile ad hoc network (MANET) formed by devices such as smartphones, which are carried by the E&R personnel. These mobile nodes can fail after a mechanical damage or running out of resources, especially battery. Furthermore, the MANET is prone to partitions due to interferences and node mobility. Mobility is a norm in E&R operations, where the movements of the personnel may follow very diverse patterns (mobility scenarios). Thus, communication in these scenarios might be performed through fundamentally different network paradigms, i.e. the end-to-end paradigm for Internet-based networks, or the Store-Carry-Forward paradigm for delay-tolerant networks. The distribution of available resources on nodes across the E&R area can be diverse as well. On many nodes, resources may be scarce. Furthermore, the E&R personnel may issue subscriptions that can vary with the diversity and number of required data sources. Also, the sources can have different sampling rates, leading to different amounts of data that needs to be processed. Finally, in E&R, reliable event detection is mission-critical. Missing or substantially delaying the detection of an event may result in loss of lives.

The aforementioned characteristics make it extremely important to design a CEP system that is efficient, reliable and reconfigurable. It has to be efficient due to scarcity of resources. Reliability is needed to cope with possible node failures and network partitions. Finally, reconfigurability is needed to adapt to diverse scenarios of mobility, resources and workloads, and to balance efficiency and reliability in these different contexts.

Centralized CEP is inefficient because all data from sources, including the data that does not lead to the detection of an event, needs to be sent to the central node. This can be addressed via distributed processing. The idea is to divide the complex subscription into parts, such that each part can be processed independently on a different node, as close to the source as possible. This way, irrelevant data can be filtered out early.

The central node in centralized CEP is a single point of failure. In distributed processing, this issue can be avoided via redundant processing, i.e. processing each part of the subscription on multiple nodes. If one of them becomes unreachable, the others can still detect events. There is however an inherent trade-off between reliability and efficiency because higher redundancy means also higher resource usage. We have to consider this trade-off in order to keep the performance of distributed CEP at an acceptable level.

Due to the strong diversity of scenarios, it is infeasible to use a single algorithm to reconfigure distributed CEP to be efficient and reliable in all scenarios. Therefore, diverse scenarios need to be addressed by different algorithms.

Our main contribution is the architecture of a distributed CEP middleware that can be reconfigured to a wide range of scenarios. We present a prototype implementation and show via emulation that the middleware can work well in diverse mobility scenarios. Reconfiguration of distributed CEP in so diverse scenarios has not been investigated before. A detailed

discussion of related work can be found in Section IV.

In this paper, Section II outlines the design of our distributed CEP system. In Section III, we present our implementation and the evaluation approach, and discuss the results. Section IV discusses the related work, concludes and gives directions for further research.

II. Design

The general idea of distributed CEP is as follows. The user submits a subscription via his node (the *application node*). In order to distribute CEP across multiple nodes in the network, the subscription is split into parts. This results in a *subscription tree*, where the vertices are called *partial subscriptions*. The leaves (*atomic subscriptions*) filter data from sources, and the other vertices perform some kind of data aggregation. Partial subscriptions are placed on nodes in the network, called the *processing nodes*. Each partial subscription can be placed on any node (or multiple nodes to increase reliability). In particular, to save resources, atomic subscriptions can be placed on nodes close to the data sources. Partial subscriptions are processed on the nodes they were placed on. Data flows in the direction from leaves to the root of the subscription tree. A node that receives a data tuple processes it with respect to all partial subscriptions. If an event is detected, a notification is generated. In the next processing step, the notification is interpreted as a data tuple and used as input to the parent partial subscription in the subscription tree. If the parent was placed on a different node, the tuple is sent to that node first. Parents are processed recursively until the top-level partial subscription is reached. When that happens, a complex event is detected and a notification is sent to the user. If there is a change in context, i.e. resources, network topology, mobility scenario, or the workloads, the initial placement of partial subscriptions may become suboptimal and may need to be adapted.

In order to realize the described approach, we identify the main concerns and propose a component-based architecture, such that these concerns are addressed in separate components: CEP, Communication, Placement, Splitting, Activation & Deactivation, Resource Manager, Data Store, Dispatcher. The reason behind this division is to reconfigure the different concerns independently of each other. This can decrease the overall cost of reconfiguration, except when multiple concerns have to be reconfigured at the same time.

Due to the diversity of the available resources and the workloads encountered in our application domains, it may be necessary to support multiple processing engines that trade off expressiveness for resource usage. Therefore, an important design decision is to encapsulate (within the CEP component) an existing centralized CEP engine, rather than extend a state-of-the-art distributed CEP system to fulfill our requirements. Currently, we use CommonSens [7] as the CEP engine, as it supports spatial and temporal operators and the source code is publicly available.

A separate Resource Manager facilitates the support for diverse scenarios of resource availability in other components. They consult the Resource Manager instead of requesting the information about resources independently from the operating system or from other middleware nodes. Thus, the cost of obtaining the information is decreased.

The Data Store allows to increase reliability of CEP by storing all subscriptions and data tuples received by the node, even if they are not going to be processed immediately.

The Activation & Deactivation component allows to increase the efficiency and reliability of CEP by deactivating certain subscriptions temporarily when resources are scarce on the node. After re-activation, the old processing state is restored, therefore historical data tuples do not need to be processed again.

A. Component design

We present the functional design of the middleware components (Figure 1). The arrows in the figure indicate communication between the components on a single node, and they start from the component that initiates the communication.

The CEP component consists of an existing centralized CEP engine and a buffer to address issues that arise due to distribution, e.g. out-of-order arrival of data tuples. The component receives as input partial subscriptions and data tuples. The CEP engine processes data tuples against subscriptions. The CEP component also provides information about the workloads to other components, e.g. Placement.

The main tasks of the Communication component are to enable communication among middleware nodes and to provide cross-layer information about connectivity to other components, e.g. Placement. To address diverse mobility scenarios, the component can be populated with algorithms that support different network paradigms: from traditional IP protocols for well-connected networks, to delay-tolerant protocols (e.g. epidemic routing [8] or message-ferrying [9]) for partitioned networks.

The Splitting component receives subscriptions as input. A complex subscription is split into parts. The result is a subscription tree, consisting of partial subscriptions and information about the hierarchy, i.e. the parents and children of partial subscriptions. The tree is returned as output.

The Placement component receives partial subscriptions or data tuples as input. Then, it executes a *placement algorithm* that selects the destinations where the subscriptions (or tuples) should be sent for processing. In order to make this selection, the algorithm may request cross-layer information about resources (from the Resource Manager), network (from the Communication component) and workloads (from the CEP component). The Placement component can be populated with algorithms that address diverse scenarios. The component returns a list of subscriptions (or tuples) and destination addresses.

The Resource Manager monitors resources (e.g. battery) on the local node and possibly remote nodes. It also keeps information about the known data sources. A data source is described by the following properties: i) the data source ID, ii) the names of data tuple attributes, iii) the address of the middleware node the source is connected to (the *data source node*). Other components can request this information.

The Activation & Deactivation component receives partial subscriptions as input. It requests information about resources from the Resource Manager and decides whether the subscriptions shall be processed on the local node. The subscriptions for which the decision is affirmative are relayed to the CEP component.

The Data Store receives and stores subscriptions and data tuples. They can be requested by other components.

The Dispatcher receives subscriptions (complex or partial) and data tuples as input and is responsible for relying them to the other components in the right order.

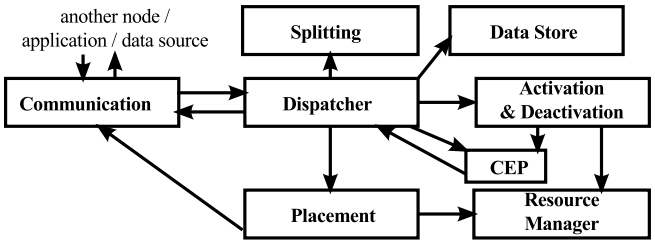

Figure 1. The component architecture of the middleware.

B. Configuration of placement

The key idea for supporting diverse mobility scenarios is to provide different placement algorithms that work well in different scenarios. From this collection of algorithms, one can be selected that is most suitable for the current scenario.

One fundamental question is: how many algorithms are needed for a potentially unlimited number of mobility scenarios? As a first step towards answering this question, we divide the space of scenarios into classes and for each class we design an example placement algorithm that works well in that class. Division into classes may follow different criteria. For simplicity, we use relative node mobility and node density. This results in four classes, where each criterion can have the value ”low” or ”high”: LM-HD (Low Mobility, High Density), LM-LD (Low Mobility, Low Density), HM-LD (High Mobility, Low Density), and HM-HD (High Mobility, High Density). We have designed three placement algorithms: *Topology Tree (TA)* for LM-HD, *Remote Areas (RA)* for HM-LD, and *One-Hop Neighbors (OHN)* for HM-HD. We omit the LM-LD class because there is no connectivity between the application node and the data sources, and therefore placement cannot be performed. In this paper, we show that it is possible to support diverse scenarios with multiple algorithms. Therefore, the presented algorithms do not need to be optimal and we do not compare their performance with existing algorithms. We proceed to describe the main ideas behind our algorithms. For each of them, we assume that the Resource Manager at the application node has complete information about data sources. Currently, placement is attempted only once by each algorithm and reconfiguration is started manually.

Topology Tree (TT). We assume that the Communication component at the application node has knowledge of the entire network topology. Placement is performed by a centralized algorithm, initiated at the application node. The key idea is to increase the efficiency of CEP by placing partial subscriptions on nodes as close to data sources as possible.

The algorithm starts by placing atomic subscriptions on the data source nodes. The remaining partial subscriptions are handled level by level, starting with the lowest level of the subscription tree and going incrementally up to the root. For each partial subscription, the following steps are executed. First, the lower-level partial subscriptions (the current subscription's children in the subscription tree) are determined, together with the addresses of the nodes where these partial subscriptions are placed. Second, the hop-by-hop routes to these nodes are retrieved from the Communication component. Third, starting from the first hop in all routes, i.e. the application node, the addresses of hops are compared. If they match, the next hop is checked. The last matching hop is the node where the partial subscription shall be placed. The reason is that this node is closest (by the hop count) to the nodes where the children are placed.

The algorithm adds to each partial subscription the addresses of the nodes where the parent subscription is placed. The Placement components at the processing nodes store those addresses and use them later to determine where to place data tuples.

Placement of data tuples is performed with a single algorithm, regardless which algorithm the partial subscriptions were placed with. The algorithm looks up the addresses of the nodes where the parent subscription was placed. These are the nodes where the data tuple shall be placed.

Remote Areas (RA). We assume at least two network partitions, and at least one message ferry. The Communication component at the application node does not know the entire network topology, and with some (or all) data source nodes it can only communicate via ferries. Placement is performed by a decentralized algorithm, initiated at the application node. The key idea is to determine which partial subscriptions can be placed directly by the application node, and which must be placed by different nodes.

The algorithm works in two stages. In the first stage, the application node places with the TT algorithm the partial subscriptions for which the destinations are in the local partition. The remaining partial subscriptions are grouped into subtrees that concern single remote partitions. The

application node places these subtrees in the appropriate partitions. Finally, the partial subscriptions that concern more than one remote partition are placed redundantly on each message ferry. The purpose of this redundancy is to increase the reliability of CEP in case a ferry fails. The second stage of placement consists of independent placements within remote partitions. In each partition, one selected node deploys the TT algorithm to perform local placement of the partial subscriptions belonging to the subtree.

One-Hop Neighbors (OHN). We assume that the application node has at least one one-hop neighbor that will at a certain time have direct connectivity with the data source nodes. Placement is performed in a centralized way. The application node places atomic subscriptions on the respective data source nodes. It places the remaining partial subscriptions redundantly on each one-hop neighbor, in order to increase the reliability of CEP in case a mobile processing node fails or moves away permanently.

III. Evaluation

We have made a proof-of-concept implementation in order to evaluate the middleware. The goal is to show how reconfiguration of the middleware components can improve the reliability and efficiency of distributed CEP in different mobility scenarios. We evaluate the middleware via emulation. The network is a MANET of IEEE 802.11b nodes. Their initial positions, links and mobility traces are simulated by the ns-3 network simulator, running on a Linux workstation. Each ns-3 node is visible to Linux as a separate TAP device. To each device, we attach an instance of Linux Containers, inside which we run the middleware and the OLSR routing daemon. We use OLSR because it is proactive and maintains on each node full information about the topology of the network partition. OLSR is the source of cross-layer information for the TT and RA algorithms.

A. Evaluation approach

Mobility scenarios. We define six scenarios (two per class). In each scenario, there are at least seven stationary nodes: six data source nodes and one application node. The communication data rate is 11 Mb/s.

Both LM-HD scenarios happen in the area of 100x60 m^2, which can correspond e.g. to a factory building. We assume the transmission range of 20 m. According to Aschenbruck et al. [1] and our own MANET testbed experiments, this range is realistic for communication indoors or in the presence of obstacles. The first scenario features a static chain network topology of 20 nodes. The second scenario features a topology of 30 nodes that move according to the RWP mobility model, at the speed of 0.25 m/s, with no pause time. All nodes are most of the time reachable from one another via unicast routes.

The HM-LD scenarios feature 11 nodes that form four network partitions. Four message ferries roam between the partitions. One data source is in the same partition as the application node. The first scenario happens e.g. in the ruins of a building. The area size is 300x60 m^2 and the partitions are 100 m apart. Nodes have the communication range of 20 m. Ferries move at the speed of 2 m/s (rescue personnel on foot). The second scenario happens e.g. in an area of an avalanche. The area size is 1500x800 m^2 and the partitions are 500 m apart. The communication range is 250 m. Ferries move at the speed of 10 m/s (e.g. snow scooters).

The HM-HD scenarios have the same area sizes and communication ranges as the HM-LD scenarios. There are respectively 23 and 8 mobile nodes. All nodes move according to the RWP mobility model with the speed of 2 m/s and no pause time. Note that, on average, the communication ranges of the nodes in the first scenario cover a lower percentage of the area than in the second scenario.

Subscriptions and workloads. It is infeasible to test the middleware with every possible subscription. Therefore, we divide the space of all subscriptions into classes and use for evaluation one subscription from each class. As the criteria for division, we use selectivity and complexity of the subscription. Selectivity is the ratio $\frac{n_i}{n_o}$, where n_i is the number of tuples that arrive as input for processing, n_o is the number of tuples that result from processing the subscription. Complexity is the number of levels in the subscription tree. For simplicity, each criterion can have the value ”low” or ”high”, yielding four possible classes of subscriptions: LS-LC (Low Selectivity, Low Complexity), LS-HC (Low Selectivity, High Complexity), HS-LC (High Selectivity, Low Complexity), HS-HC (High Selectivity, High Complexity). For the evaluation, we use four subscriptions (CS1-CS4 respectively), one from each class. The subscription complexity is 2 for CS1 and CS3, and 4 for CS2 and CS4. Selectivity is determined by different predicates: low selectivity by the OR predicate, and high selectivity by the combination of the AND predicate and temporal constraints. For each subscription, we use a workload that allows to detect 5 complex events. Each data source sends data at the rate of 1/s.

The experiment. The application sends the complex subscription. We let the partial subscriptions propagate to the processing nodes. Then, the data sources send data. For subscriptions CS1 and CS2, this takes 10 s, for CS3: 201 s, and for CS4: 310 s. After the data is sent, we wait until all events are detected or until 2800 s has elapsed. There are 288 configurations of: mobility scenarios (6), placement algorithms (4), subscriptions (4), and tuple buffer sizes (3). We run 5 iterations of each.

Metrics. We use two metrics to measure the reliability of CEP. The first is the *number of true positives* (NTP), i.e. the number of correctly detected events. If the application node receives more than one notification of a single event, the excess number is expressed by the *number of duplicates* (ND). We measure efficiency using the *overhead* (OV)

metric, i.e. the total size (in bytes) of the messages sent or forwarded by middleware nodes. For clarity, OV does not include beacons from OLSR and epidemic routing. The *notification delay* (D) depicts both reliability and efficiency. It is measured for the first notification of each event and is defined as the difference between the time the last relevant reading was received from a data source, and the time the event notification was received by the application.

B. Results

The evaluation shows that distributed CEP works reliably and efficiently when we use the placement algorithm tailored for the current scenario class. Applying a different algorithm generally leads to missing events or increased traffic in the network. In a few configurations however, more than one placement algorithm performs well.

Figure 2 contains the average values and tolerance intervals of NTP, ND, D and OV for one mobility scenario from each class, under the assumption that out-of-order data tuples are buffered indefinitely. The confidence intervals are computed for confidence level of 95%.

The OHN algorithm works in all configurations, however it works best in the HM-HD scenarios. In the LM-HD scenarios, OV is much higher (3-20 times, depending on the subscription) than with the TT algorithm due to redundant processing. However, we observe redundancy leads to lower D (5-30 times) in the scenario with slow mobility. In this scenario, the network topology changes occasionally (on average, once every 2 s) and packet loss may occur when sending data tuples, especially via multi-hop routes. Redundancy means that tuples may be sent via multiple routes. Some of them may be stable, allowing a delivery with few retransmissions and therefore a low D. This observation shows that the mobility scenario is in fact near the border between the LM-HD and HM-HD class. Increasing the speed of nodes pushes the scenario into the HM-HD class, where the TT placement algorithm no longer works. In the HM-LD scenarios, OV is high with the OHN algorithm, due to the fact that one of the one-hop neighbors of the application node is stationary. Since several partial subscriptions are processed on that node, many data tuples need to be sent to it from remote partitions. In the HM-HD scenarios with the OHN algorithm, the unstable network and high node density lead to a high OV that varies significantly across different runs of the experiment.

The RA algorithm works in LM-HD and HM-HD, but only for subscriptions with low complexity (CS1 and CS3). The reason that it does not work for subscriptions with high complexity is that the algorithm attempts to place some partial subscriptions on message ferries, which only exist in the HM-LD scenarios.

As expected, duplicate notifications are encountered in configurations where the top-level partial subscription is processed redundantly (the OHN algorithm). ND for subscriptions with high complexity is significantly higher (5-10 times) than for subscriptions with low complexity. The reason is that in the former case there are more partial subscriptions. When processing a partial subscription results in a data tuple, this tuple is sent to the nodes where the parent subscription is placed. Processing each replica of the parent subscription may result in a further data tuple. This procedure continues recursively until the root of the subscription tree is reached. Therefore, the number of generated data tuples grows exponentially with the subscription complexity.

In centralized CEP ("C" in Figure 2), performance depends heavily on the number of data tuples to send. If it is high, as for selective subscriptions CS3 and CS4, the packet loss due to network partitions may lead to retransmissions, thus increasing OV and D. Some tuples are not delivered during the experiment, leading to missing events, e.g. NTP=4 for CS2 in the first LM-HD scenario. For CS4 in HM-HD scenarios, the number of simultaneous transmissions causes saturation of the simulator and inability to deliver enough tuples to detect any event (i.e. NTP=0). For CS2, centralized CEP has significantly lower OV (up to 11 times) than with the RA or OHN algorithm, however D is 1.5-2 times higher. The reason is that less data needs to be sent through the network, but there are fewer connectivity possibilities because the data is sent directly to the application node.

In the presented experiments, we have assumed that out-of-order data tuples are buffered indefinitely. If the buffer holds tuples for a shorter time, NTP drops for selective subscriptions (e.g. NTP=1 for CS4 in the first HM-LD scenario, RA algorithm, buffer time = 0 s). The reason is the semantics of the "AND" predicate and predicates with temporal constraints. They require "true" tuples with each attribute, and with specific timestamps. In the case of non-selective subscriptions (CS1, CS2), decreasing the buffer length causes a significantly ($>$ 100 times) shorter D, however ND could increase 8 times. This happens because the "OR" predicate used in the subscriptions requires "true" tuples with only one attribute. If tuples with different attributes but identical timestamps arrive at different times, a single complex event is detected more than once.

IV. Conclusion and future work

In this paper, we introduce a distributed CEP middleware that can be reconfigured to diverse scenarios of mobility, resources and workloads, and can balance the efficiency and reliability of CEP in these scenarios. The middleware is component-based, which allows to reconfigure several concerns of distributed CEP (e.g. communication, placement of subscriptions and data tuples) independently. In the evaluation of our proof-of-concept implementation, we demonstrate reconfigurability to diverse mobility scenarios.

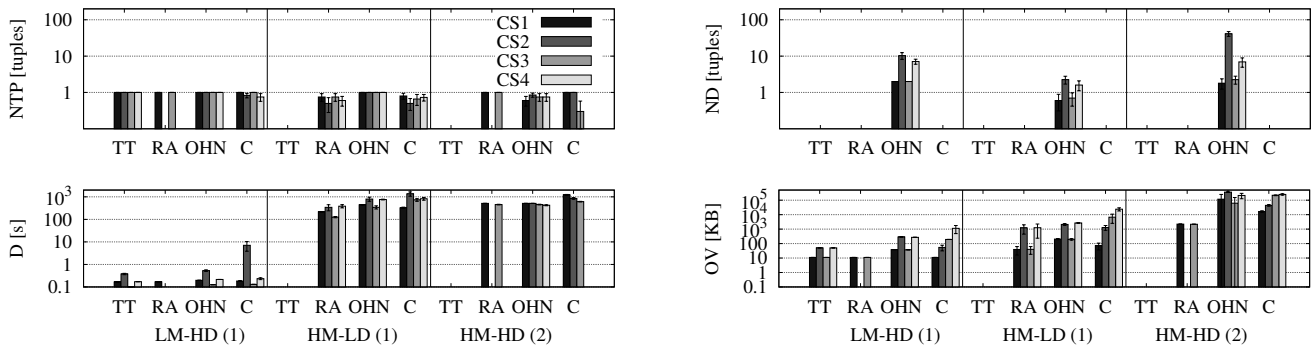

Figure 2. The results for selected configurations in the following mobility scenarios: LM-HD (1), HM-LD (1), HM-HD (2).

Reconfigurable and adaptive solutions for distributed CEP (or in-network processing) and distributed data stream processing have been researched in the context of WSNs [3] [4], MANETs [2], infrastructure networks [6] [5], and combinations of those [10]. The focus is on adapting [3] [2] [6] [5] or reconfiguration [10] of placement of partial subscriptions. GINSENG [4] adapts the configuration of processing parameters, e.g. the sampling rates of sources. Our work investigates reconfiguration of placement, however the architecture is designed to support also other aspects of distributed CEP. All approaches focus mainly on efficiency of distributed CEP. Zhu et al. [10] models the event notification delay and energy usage during CEP. The other approaches monitor and react to changes in resources [5], network (delay [4] [6] [5], bandwidth [6] [5], loss [4], topology [2]), and the data tuple rates [3]. The approach of Avvenuti et al. [2] is the closest to our work, however we go beyond it by supporting network topologies in diverse mobility scenarios. The reliability requirement is addressed proactively (by placement on backup nodes) [3] or reactively (placement on a new node after a node failure) [5]. The former is less efficient but more reliable than the latter if nodes can fail unexpectedly. Our approach can support both, and allows to balance reliability and efficiency. There exists a reconfigurable approach [10] that can send partial subscriptions and data tuples either in peer-to-peer manner over a MANET or unicast over an infrastructure network. The proposed performance model uses a single partial subscription, and investigates only a scenario with random waypoint mobility. Instead, our approach supports arbitrary subscriptions and mobility scenarios.

The contribution of this paper is a first step to enabling CEP in new application domains like E&R. Our ongoing research follows two main directions. First, we aim to perform reconfiguration automatically. The key idea is to detect relevant changes in the scenario and then reconfigure the appropriate middleware components. We plan to investigate alternative placement algorithms to the ones presented in this paper. Second, we will research the benefits of reconfiguring other middleware components, e.g. Splitting or Activation & Deactivation.

ACKNOWLEDGMENT

The work was funded by the SIRIUS project (ref. 2009/784) of the Norwegian Research Council.

REFERENCES

[1] N. Aschenbruck, R. Ernst, and P. Martini. Indoor mobility modelling. In *GLOBECOM Workshops*. IEEE, 2010.

[2] M. Avvenuti, A. Vecchio, and G. Turi. A cross-layer approach for publish/subscribe in mobile ad hoc networks. In *Proc. of the 2nd MATA*. Springer-Verlag, 2005.

[3] B. J. Bonfils and P. Bonnet. Adaptive and decentralized operator placement for in-network query processing. In *Proc. of the 2nd IPSN*. Springer-Verlag, 2003.

[4] Z. Jerzak, A. Klein, and G. Hackenbroich. GINSENG data processing framework. In *Reasoning in Event-Based Distributed Systems*. Springer-Verlag, 2011.

[5] G. G. Koch, B. Koldehofe, and K. Rothermel. Cordies: expressive event correlation in distributed systems. In *Proc. of the 4th DEBS*. ACM, 2010.

[6] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *Proc. of the 22nd ICDE*. IEEE, 2006.

[7] J. Søberg, V. Goebel, and T. Plagemann. Commonsens: Personalisation of complex event processing in automated homecare. In *Proc. of the ISSNIP*. IEEE, 2010.

[8] A. Vahdat and D. Becker. Epidemic routing for partially-connected ad hoc networks. Technical Report CS-200006, Duke University, 2000.

[9] W. Zhao and M. H. Ammar. Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks. In *Proc. of the 9th FTDCS*. IEEE, 2003.

[10] X. Zhu, B. Xu, and O. Wolfson. Spatial queries in disconnected mobile networks. In *Proc. of the 16th SIGSPATIAL*. ACM, 2008.