An Innovative Architecture for Context Foraging

Vassileios Tsetsos
Pervasive Computing Research Group
Dept. of Informatics and Telecommunications
University of Athens, Greece, Panepistimiopolis, Ilissia, 15784
b.tsetsos@di.uoa.gr

Stathes Hadjiefthymiades
Pervasive Computing Research Group
Dept. of Informatics and Telecommunications
University of Athens, Greece, Panepistimiopolis, Ilissia, 15784
shadj@di.uoa.gr

ABSTRACT

Nomadic computing is a term for describing computing environments where the nodes are mobile and have only ad hoc interactions with each other. Evidently, context aware applications are a key ingredient in such environments. However, nomadic nodes may not always have the capability to sense their environment and infer their exact context. Hence, applications carried by the nodes will not be able to execute properly. In this paper, we propose an architecture for collaborative exchange of contextual information in an ad hoc setting. This approach is called "context foraging" and is used for disseminating contextual information based on a publish/subscribe scheme. We present the algorithms required for such architecture along with the dynamic event indexing techniques used by the system. The efficiency of the suggested approach is assessed through simulation results. Our proposal is investigated and implemented in the context of the ICT IPAC Project.

Categories and Subject Descriptors

H.3.4 [Systems and Software]: *Current awareness systems (selective dissemination of information--SDI),* C.2.4 [Distributed Systems] *Distributed applications*

General Terms

Algorithms, Design, Experimentation

Keywords

Collaborative sensing, nomadic computing, publish-subscribe

1. INTRODUCTION

Nomadic Computing (NC) is the term referring to highly variable computing and communication environments which serve nomadic users [1]. Key characteristics of nomadic users are:

- frequent relocation (following various mobility patterns),
- use of low-power and low-capability devices, and
- use of mobile and adaptive applications.

The communication part of such environments resembles mobile ad hoc networks (MANETs), but their added value goes beyond ad hoc communications. Specifically, one key aspect is the provisioning of applications that really adapt to the ever changing environment in a seamless way. This is an approach also taken in Pervasive and Ubiquitous Computing environments. In this paper we propose a novel framework for implementing context awareness in a nomadic ecosystem through efficient collaboration between its nodes. Specifically, mobile nodes request context they cannot sense with their own sensors from other nodes in their vicinity. This paradigm of context information dissemination is referred to as "Context Foraging" since it resembles the concept of Cyber Foraging [2].

Efficiency in such environments is very important for two basic reasons:

1. Nodes are typically devices with limited resources (small or no user interfaces, battery lifetime constraints, limited memory and processing power, etc.). Hence, applications should be very careful regarding the resources they utilize.
2. There is no easy way to implement reliable and efficient networking protocols (even ad hoc routing protocols) so it is common to exploit variations of broadcasting. Hence, applications should be very careful as to the volume of data they disseminate.

From the above, it becomes apparent that every service or application deployed in such environment should minimize the network traffic it generates.
In the case of context foraging, an additional requirement is the ability of the nodes to detect context changes. This is very important since detection of such changes may affect the behavior of the applications. Such detection is performed in two phases. Firstly, a node collects all, or as much as possible, sensor readings of interest. Then, a reasoning process takes place locally and context changes are inferred. In this paper we mostly deal with the first phase. However, the system model presented in Section 2, describes also the elements that enable the second phase. In fact, knowledge representation and reasoning techniques in mobile and embedded computing is an area of ongoing research.

The rest of the paper is organized as follows. Section 2 describes the nomadic computing environment and the proposed architecture for Context Foraging. Some assumptions regarding the nodes, the network and the application modeling are also mentioned. In Section 3 we describe in detail the algorithms involved in the proposed architecture. A performance evaluation through simulations that validates the efficiency of the system is presented in Section 4. Finally, the paper concludes with related work and directions for future research.

2. SYSTEM MODEL AND ASSUMPTIONS

Each node has a Context Foraging layer in its stack which comprises just one service of a more generic middleware (see Figure 1). This component is clearly independent from the upper (application) and lower (communication) layers. However, some basic assumptions are made for these layers, as described in the following paragraphs.

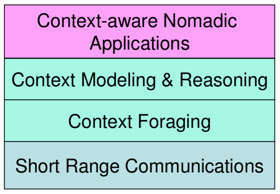
Context-aware Nomadic Applications
Context Modeling & Reasoning
Context Foraging
Short Range Communications

Figure 1. Architecture of a node

2.1 Application and Context Modeling

Regarding the applications, we assume that they are written with the aid of declarative languages (i.e., rule languages). Writing context-aware and situation-aware applications by following a declarative (knowledge-based) approach is a widely-adopted paradigm with many advantages as reported in [4]. In the following paragraphs we define the knowledge representation elements that are used in the framework.

Context is represented through an ontology with two types of relationships between its concepts: subsumption (is-a) and part-of relationships (part-of relationships are mainly used for spatial context representation). A sample context ontology is shown in Figure 2a. Other similar ontologies can be found in [4][5][6].

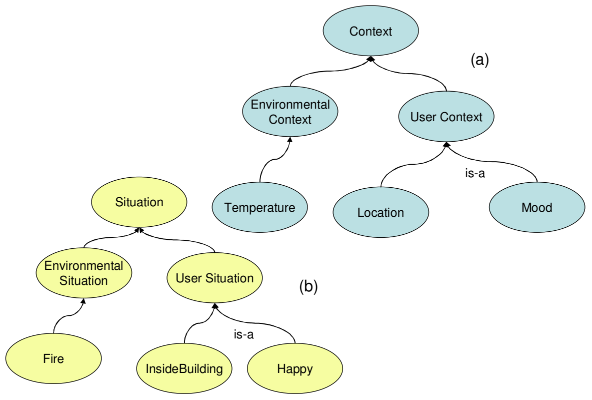

Figure 2. a) a sample context ontology, b) a corresponding situation ontology

Let C be the set of all context classes (types). Let LC_N be the set of context classes that are supported by node N, i.e., its "local" context. In other words, node N has all the required sensors to produce values for these context classes. Each class C_i has a relationship (aka, property) val_i, the value set of which (also called "range" in ontology terminology) is denoted as $R(C_i)$. Moreover, each C_i has a default spatial validity value SV_{Ci}, that reflects the range within which the values for a context type are regarded valid (i.e., they are expected to be approximately the same, see also Definition 3).

A context class may not always be a one-to-one correspondence to a specific sensor. Higher level context classes can be described through conjunctive rules of the form:

$(val_1 \text{ op } V_1) \wedge (val_2 \text{ op } V_2) \wedge \ldots \wedge (val_m \text{ op } V_m) \rightarrow C_i$

where $i>m$, $C_i \in C$, $V_i \in R(C_i)$, op $\in \{>, <, =, <=, >=\}$

Definition 1. ***Context Request (CReq)***

An *atomic* context request generated by a node is defined as: $CReq := val_i \text{ op } V_i$, $V_i \in R(C_i)$, op$\in \{>, <, =, <=, >=\}$

A *composite* context request generated by a node N is defined as a set of atomic requests (all atomic requests are treated separately during request handling and not as a conjunctive experssion):

$CReq' := \{val_1 \text{ op } V_1, \ldots , val_i \text{ op } V_i, \}$, $V_i \in R(C_i)$, op$\in \{>, <, =, <=, >=\}$

Definition 2. ***Context Response (CRes)***

A context response is a set that contains one or more (context class, context value) pairs:

$CRes := \{val_1 = V_1, \ldots, val_i = V_i\}$, $V_i \in R(C_i)$.

A context response has also a spatial validity parameter which is the maximum of the individual spatial validity values included in the response.

Definition 3. ***Spatial Validity of a Context Request (SV_{CReq})***

The range within which the context values included in the request are valid/useful for the requesting node. Such range may depend on the degree of locality of the phenomenon and/or other application characteristics. If we assume that we adopt circular spatial modeling, then SV_{CReq} is the radius of a circle, with center the current position of the Context Requestor. This circle includes all nodes that can provide valid values for the *CReq* context classes. If SV_{CReq} is explicitly defined by the application for a specific request then it applies to all involved context classes. Otherwise, for each atomic *CReq*, the default spatial validity value (SV_{Ci}) applies. Note that similarly to requests, context responses also have a spatial validity that controls their dissemination.

Definition 4. ***Temporal Validity of a Context Request (TV_{CReq})***

It is the time period, measured from the moment the initial request was issued, until the moment the request is not regarded valid. This value is a measure for the context freshness. Alternatively one can think of TV_{CReq} as the time interval between two retransmissions of CReq.

2.1.1 Situation-aware Computing

Situations can be regarded as higher-level descriptions for the current activities and contextual status of nodes/users, and they affect the actions that the applications should take.

Let S be the set of all situation classes (types). If we assume that we are interested in situations regarding users, nodes, and the environment, then S can be divided to three subsets, S_u, S_n and S_e, respectively. Situations can be described through an ontology, too (see Figure 2b), and for their classification specific rules apply.

Definition 5. ***Situation Classification Rule (SCR)***

A rule that defines, either through necessary (or necessary and sufficient) conditions, a situation of a user, a node, or the environment. The general form of such rule is:

$SCR_i := S_1 \wedge \ldots \wedge S_k \wedge (val_1 \text{ op } V_1) \wedge (val_2 \text{ op } V_2) \wedge \ldots \wedge (val_m \text{ op } V_m) \rightarrow S_i\,(SV_{Si}, TV_{Si})$

where i>k and $\rightarrow$ denotes that the conditions in the rule body are necessary ($\equiv$ is used for necessary and sufficient conditions). SV_{Si} is the spatial validity of SCR_i. Only S_i that are subclasses of S_e can have spatial validity and in this paper we deal only with such type of situations. TV_{Si} is the temporal validity of SCR_i.

Each condition in the rule's body can be evaluated to one of the following *status values*:

- *Unknown*: the context class of the condition is not in LC_N
- *True*: the context class of the condition is in LC_N and the condition is satisfied by the current context value
- *False*: the context class of the condition is in LC_N and the condition is not satisfied by the current context value.

The conditions that are either true or false are called *local conditions* while the conditions that participate in the rule's context request are called *remote conditions.* The conditions of a rule SCR_j with status *unknown* will eventually form a Context Request $CReq_j$ which inherits the SV_{Sj} and TV_{Sj} values.

Finally, each SCR has a *"trigger level"*. This level is a value indicating how closely to firing the rule is (i.e., the number of satisfied local conditions divided by all local conditions). The trigger level assumes the value 1 when all the local conditions are satisfied and 0 when no local condition is satisfied. The usage of the trigger level is described in Section 3.5.

As already stated, situations are used for determining the application actions in a declarative way.

Definition 6. ***Action Rule[1] (AR)***

A rule that defines the actions that should be triggered if all the conditions in its body are satisfied. For example, such actions could be method invocations if procedural attachments are supported by the rule language employed. The conditions involve situations and, thus, indirectly context values. The general form of such rule is:

$AR_i := S_1 \wedge S_2 \wedge \ldots \wedge S_m \rightarrow$ SomeAction (SV_{ARi})

where SV_{ARi} denotes its spatial validity.[2]

2.1.2 Example

The following example better explains the aforementioned definitions. Let us assume that a node N1 has only a temperature sensor and the action rule:

Fire → BroadcastAlert (100)

meaning that if fire is detected within a range of 100 space units then the node should broadcast an alert. Let us further assume that the situation *Fire* is defined through the SCR:

Temperature>80 ^ Humidity<10 → Fire (100, 10)

, where 100 (SV_{CReq}) is the range within which humidity values are regarded valid. The value 10 (TV_{CReq}), defines how often context requests for humidity values will be retransmitted to nodes that have humidity sensors. Hence, N1, every 10 time units, will issue a CReq of the form:

Humidity<10

and this request will reach only the nodes in the range of 100 space units. When a node with a humidity sensor satisfies the condition of the CReq (e.g., humidity is 5 degrees) it will broadcast a CRes:

Humidity=5

which will also reach the nodes within range of 100 space units. Note that the terms Fire, Temperature and Humidity should be defined in the ontologies of Figure 2.

[1] We just deal with SCRs because ARs are just another abstraction layer. However we mention them for completeness purposes.

[2] Temporal Validity is not defined for action rules since it mainly concerns lower level rules (i.e., SCR)

2.2 Communications and Other Assumptions

For the communication layer, the main assumption is that we do not have any means of high-level networking protocols and information exchange is performed through (some variant of) a broadcasting scheme. The broadcasting is performed based on short range communications (e.g., ZigBee, IEEE 1609 WAVE, IEEE 802.11 ad hoc mode) and is not global. Several efficient variants of broadcasting have been proposed in the literature such as epidemic information dissemination [7], gossiping [8], etc. Another assumption is that some special flags in packet headers, indicating the types of messages used by the scheme (i.e., context requests, context responses), can be set.

Finally, the proposed framework assumes an application environment with the following characteristics:

- all nodes are mobile,
- there is no assumption as to how many nodes carry sensors or other context detecting devices,
- all nodes are willing to collaborate so that their respective context-aware applications are executed in an optimal way,
- each node can estimate its location (through some type of location sensor, e.g., GPS),

3. FRAMEWORK ARCHITECTURE

3.1 Context Foraging Workflow

There are three roles that a node can assume:

1. Context Requestor (CR), if the node requests context values from other nodes.
2. Context Relay (CRel), if a node does not have the sensors required by a context request or is not interested in the context response contents. CRels just forward messages that they have not forwarded before.
3. Context Provider (CP), if it can sense and send to other nodes some type of context.

In each node there is an instance of the context ontology and the situation ontology along with the SCRs. Each application, upon deployment, contributes its action rules to the knowledge base of the node. If the Action Rules of some application contain situations with *remote conditions* then the respective context requests are created and disseminated. Once a CReq reaches a CP then it is registered as an event filter in it. When an event satisfying a filter is triggered by the CP sensor values, a CRes is disseminated in the network.

The overall scheme a variant of a content-based publish/subscribe scheme, adjusted appropriately for the nomadic computing paradigm. Specifically, each publisher (CP) is also a broker, since publish/subscribe systems for networks with high topology change rate should store the event interests very close to the information sources. Moreover, some type of subscription covering (aggregation) is performed, extending that found in content-based networking schemes (e.g., [13]).

3.2 Context Requestor Design

A CR can collect context values required by its knowledge base (Action and Situation Classification Rules) and not provided by its own capabilities. Context foraging adopts the following principles:

- The context values received through Context Responses are spatially valid.

- The context values are fresh, i.e., context values are received as soon as they are generated, ensuring a good sensitivity in detecting situation updates. No context-caching is used.

Moreover, the two non-functional, efficiency requirements mentioned in Section 1 apply, too. Listing 1 presents the algorithms implemented in the CR. The *requestContext* algorithm can be executed periodically (for each SCR rule) or whenever some rules are "close to fire" (see Section 3.5). The *responseHandler* algorithm is executed whenever a new CRes is received by the CR.

function requestContext

Input: SCR: an array of all SCRs

```
1: while true
2:   for i=1 to SCR.length do
3:     if SCR[i].hasRemoteConditions()
4:     then sendRequest(SCR[i].remoteConditions);
5:   end for
6:   rescheduleRequest(SCR[i].remoteConditions, SCR[i].TV);
7: end while
```

function responseHandler

Input: CRes: the array that contains all conditions of the Context Request, SuppClasses: the array of all context classes supported by the node

```
1: if isSpatiallyValid(CRes) then
2:   if CRes.classes not in SuppClasses then
3:     assert(CRes);
4:     fireRules();
5:   forward(CRes);
```

Listing 1. Request Formation and Response Handling

3.3 Context Provider Design

Each node that can assume a context provider role has an index data structure used for two purposes: a) as a registry of all event filters received through context requests, and b) as a mechanism that matches events (new sensor values) with event filters

The main idea is that context requests will be registered (with their respective timeouts) in this index. The sensor stream will be also fed into this index so that sensor values that match some filters generate events that are disseminated through the network. Hence, this index will act like a message forwarding engine that is typically found in content-based network routers [13]. Due to size limitations we do not describe this event generation process, which is rather straightforward. The only special point is that the context responses are also assigned a Spatial Validity value which is the maximum of the spatial validities included in the response elements (so that all relevant requestors can be reached).

The structure of this index is depicted in Figure 3. In this index, the arrays *FILTERS* and *EQUALS* are sorted. Specifically the *FILTERS* array contains the context conditions (also called event filters or subscriptions) received through incoming CReqs and is sorted in descending order for the '<' and '<=' operators and in ascending order for all other operators. Sorted arrays are used because the readings in this index (new context values, generated by sensors) are expected to considerably outnumber the updates (new event subscriptions). The *EQUALS* array (also stores contextual conditions) constitutes an optimization in order to avoid unnecessary filters. For example, for the event "contextClass[1] = 22", we do not create an additional filter in the *FILTERS* array of the '=' operator since there is an overlap with an existing filter ("contextClass[1] >20"). Finally, the *timeout* values are used for deciding the expiration of the filters and the *SV* values for setting the spatial validity parameter in the context response. Listing 2, describes how this index is used during request handling in CPs.

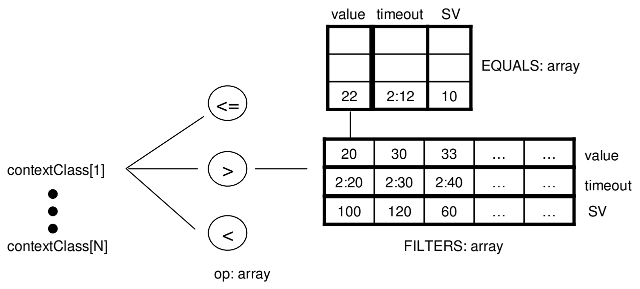

Figure 3. The index of event filters

function requestHandler

Input: CReq: the request object, SuppClasses: the array of all context classes supported by the node

```
1: for i=1 to CReq.conditions.length do
2: begin
3:   if isSpatiallyValid(CReq.conditions[i])
4:     if CReq.conditions[i].contextClass in SuppClasses then
5:       subscribeEvent(CReq.conditions[i]);
6: end;
7: sendRequest(CReq);
```

function subscribeEvent

Input: CReq: an atomic context request, index: the index of Figure 3.

```
1: if CReq.op not in index[CReq.contextClass].op then
2:   newOp = index[CReq.contextClass].op.add(CReq.op);
3:   newOp. FILTERS.add(CReq.value, CReq.timeout, CReq.SV);
4: else
5:   filter = index[CReq.contextClass].op[CReq.op]. FILTERS;
6:   filter.addFilter(CReq.value, CReq.timeout);
```

function addFilter

Input: CReq: an atomic context request, index: the index of Figure 3.

```
1:  currentValue =
index[CReq.contextClass].op[CReq.op].FILTERS.getValue(1);
2:  if CReq.op = '>' and currentValue > CReq.value or
    CReq.op = '>=' and currentValue >= CReq.value or
    CReq.op = '<' and currentValue < CReq.value or
    CReq.op = '<=' and currentValue <= CReq.value then
3:  then this.addFirst(CReq.value, CReq.timeout, CReq.SV);
4:  else if CReq.value in this.value then
5:    this.rescheduleTimeout(CReq.value, CReq.timeout, CReq.SV);
6:  else
7:    this.add(CReq.value, CReq.timeout);
8:  if CReq.op = '=' and CReq.value >
    index[CReq.contextClass].op['<'].this.getValue(1) and CReq.value <
    index[CReq.contextClass].op['>'].this.getValue(1) then
9:    index[CReq.contextClass].op['='].filters.add(CReq.value,
      CReq.timeout, CReq.SV);
10: else
11:   this.addEqual(CReq.value);
```

Listing 2. Request Handling in Context Providers

Since, in nomadic computing nodes relocate frequently, we do not want to store all the event filters infinitely in the index since the spatial validity of the respective events (i.e., context responses) is affected by the nodes' mobility (for both types of nodes, CR and CP). For that purpose, we use timeouts for the filters and the algorithm in Listing 3 handles their expiration. These timeouts are scheduled in a typical job scheduler. What this algorithm does, is to remove the expired filters from the index and the job scheduler but also keep the index in a consistent state. Specifically, if there is an *EQUALS* array linked to a *FILTERS* element, then it either links it to another element or registers its elements as new filters under the '=' operator (the second case holds if the "contextClass[1]>20" filter in Figure 3 expires). As shown in line 6, upon removal of a *FILTERS* element, its associated *EQUALS* array, if any, is linked to the left element. This happens due to the sorting of the *FILTERS* arrays.

```
Algorithm filterExpired
Input: expFilter: the expired filter, index: the index of Figure 3
1: if expFilter.op != '=' and expFilter =
   index[expFilter.contextClass].op[expFilter.op].FILTERS.get(1) then
2:   if expFilter.equal is not empty then
3:     for all i in EQ do
4:       index[expFilter.contextClass].op['='].FILTERS.add(i.value,
i.timeout, i.SV);
5: else if expFilter.equal is not empty then
6:     move(expFilter.equal); //move the equal array to the left filters item
7:  FILTERS.remove(expFilter);  //removes it from the filters array and the
scheduler
```

Listing 3. Expiration of event filters

3.4 Some special cases

Given the above algorithms, a low overhead event-driven collaborative sensing scheme can be implemented. The advantages are that no continuous polling, which is very resource demanding is used, and useless event filters are removed by the filter expiration mechanism. Moreover, this scheme copes well with the following problematic cases:

A) The CR leaves the region after it has sent a CReq: In this case the CP nodes transmit context responses just until the CR's event filters expire.

A.1) Even worse, some context responses to this CReq, which are spatially invalid, reach the CR when it is far from its initial position: Spatial validity is set for context responses (since we assume all nodes know their location).

B) The CPs with registered event filters go away from the CR: They transmit context responses to their new neighborhood just until their timeouts expire. After all, the new neighbors may be also interested in the context values included in these responses.

However, one possible case that may arise and if not handled may affect the *sensitivity* of the system is the following:

A CR disseminates a CReq for a context class *C* with a long time validity (late expiration). We remind that long time validity is desirable because it reduces the traffic and the utilization of other node resources used during CReq dissemination. We assume that no neighbor of the CR is a CP for this context class. Then, just after the CReq dissemination, a new node appears that can fulfill this request. In this case, the new node will be able to sync with the other nodes only after the expiration of the previous request and the retransmission of a new one.

To deal with this case, each node could possibly retransmit the CReqs received by others or generated by itself to all of its one-hop neighbors upon change in its one-hop neighborhood topology. However, we did not include this functionality in the scheme since some first simulations showed that it introduces considerable communication overhead.

3.5 Lazy Context Requesting

A last mechanism proposed for the Context Foraging architecture is the "Lazy Context Requesting" and it refers to how and when CReqs should be generated. Specifically, a situation may never happen or may happen only very seldom. In such case, exploiting the context foraging scheme described in the previous section is not probably the most efficient solution. For example, let us assume the following situation rule:

(Temperature>80)^(Humidity<20)^(Smoke=true) → Fire

and that the node can provide Humidity and Smoke values (these are the local conditions of the rule). Obviously, it is a waste of resources to initiate the context foraging process when Smoke=false and Humidity>=20 since the fire event will never be detected even if Temperature values, received from neighbor nodes, are higher than 80 degrees.

In order to optimize the context foraging process the CReqs (i.e., event subscriptions) are not sent in a static but in a *context-aware* way. The main concept is that a node sends CReqs derived from a situation classification rule to the network only when its own context values satisfy the local conditions of the rule. Hence, there is a meta-rule engine that monitors the *trigger level* of the SCRs. Once a rule's trigger level passes a threshold, then the corresponding CReq is broadcast (i.e., the *sendRequest* function is called). The threshold involved in this mechanism may depend on the criticality of the rule (i.e., critical rules have lower thresholds) or other background knowledge we have for the rule so that premature context requesting can be performed, if necessary.

4. PERFORMANCE EVALUATION

In order to assess the performance of the proposed scheme we ran several simulations, where we compared it to an alternative scheme that satisfies many of the problem requirements. This scheme (called Context Polling, or CPol for brevity) is simpler and with some inherent limitations but, in our opinion, seems to be one of the most promising alternatives for the specific domain.

In CPol, each Context Requestor periodically sends CReqs to other nodes. If the Context Providers, satisfy some (parts of) requests, they immediately send their context responses and discard the CReq. Hence, the scheme is totally stateless, which is also one of its key advantages. However, the main drawback is that as the period increases, the CRs cannot detect context changes of interest. On the other hand, as will be shown in the following section, if the period is set to one time unit then the communication overhead is too high.

4.1 Simulation Setup

In order to assess the performance of the context foraging scheme we ran several simulation scenarios, using the following metrics:

1) Number of exchanged messages (#Msg). It involves all CReq, CRes and their corresponding forwards.
2) Average Situation Detection Ratio (ASDR). The average SDR over all Context Requestors. Situation Detection Ratio for a CR i is defined as:

$$SDR_i = \frac{A}{B}$$

where A: Number of SCRs fired by node i, B: Number of SCRs that should be ideally fired by node i. The parameter (counter) B is increased by one each time a combination of sensor values that would trigger a SCR of the node i was observed within the area where the rule is spatially valid.
Other typical metrics in publish/subscribe systems such as delivery latency are not useful in our case, because all events are local to the requesting node. Hence, the expected delay of a context response is known a priori and it depends on the spatial validity assigned to the corresponding request.

The first part of the simulation compares the performance of the two schemes, CFor (for Context Foraging) and CPol. The second part tries to evaluate the use of the lazy context requesting mechanism. Hence, in Part B we ran the set of experiments B1 and B4 that are the same as the A1 and A2 but *with lazy context requesting enabled*.

Table 1 summarizes the default setup parameters (for both parts) while Table 2 presents the parameters for the specific experiments.

Table 1. Simulation Setup – Default parameters

# of nodes	100
Simulation time	200*
Mobility model	Random waypoint (see [3]): Maximum pause time: 20 Min speed: 0
# of SCR per CR	2
SV of SCRs	110
Communication range	50
Environment dimensions	500x500
Trigger level for Lazy requesting	0.66 (i.e., 2 out of 3 conditions)
# of avg one-hop neighbors	~3

*All times are expressed in time units

The following SCRs were assigned to the CRs for Part A:

Rule	# of CRs
SCR_1: (Temperature>80) ^ (Humidity <20) → Event 1 SCR_2: (GasA>40) ^ (GasB>50) → Event 2	1/3 of CRs
SCR_3: (Humidity <20) ^ (Smoke=true) → Event 3 SCR_4: (GasB>50) ^ (Smoke=true) → Event 4	1/3 of CRs
SCR_5: (Temperature>80) ^ (Smoke=true) → Event 5 SCR_6: (GasA>50) ^ (Smoke=true) → Event 6	1/3 of CRs

This division of the CRs to three sets, with rules that do not fully overlap in terms of context classes, was made so that not all nodes produce the same context requests. If that was the case, the requests would result in being "global" and would bias the simulation results in scenarios with mobile nodes.
The two rules in Part B that were shared by all CRs are:
SCR_7: (Temperature>80) ^ (Humidity <20) ^ (Smoke=true) → Event 1
SCR_8: (GasA>40) ^ (GasB>50) ^ (Smoke=true) → Event 2

The CRs in Part A are not CPs (i.e., do not have any sensors). In Part B, the CRs have Humidity, Smoke and GasB sensors. In both parts, the CPs (excluding the CRs in Part B) have one sensor each and each context class is provided by an equal number of CPs, on average. The sensor values were generated once and used in all runs of an experiment. The sensor values are random and the difference of two subsequent values follows a Normal distribution N(0,1).

Table 2. Simulation Setup – Experiments

Part A	
Experiment A1	
Mobility model	Max speed: 0, 1, 2, 4, 10, 20
# of CR	40
# of CP	40
Experiment A2	
Mobility model	Max speed: 2
# of CR	40
# of CP	40
SV of SCRs	30, 50, 70, 90, 110
Part B	
Experiment B1 (same as A1)	
Experiment B2 (same as A2)	

4.2 Simulation Results

A first and rather trivial observation is that if we set the time validity of all SCRs in the Context Foraging scheme equal to one time unit, the scheme becomes identical to CPol with period one time unit, in terms of ASDR. However, the CFor scheme transmits much fewer messages due to the response aggregation that is performed through the index.

Figures 4, 5 summarize the results for experiment A1. Two variants of Context Foraging (CFor) and two variants of Context Polling (CPol) are compared (the numbers after the scheme names denote the temporal validity of the context requests). All schemes except for CPol2 have similar event detection capability (ASDR). However, the CFor schemes exchange considerably fewer messages. Moreover, the message overhead of CPol increases exponentially, whereas that of CFor linearly (Figures 4 and 6).

Increasing the TV for CFor does not affect significantly the ASDR values and provides substantial message reduction. However, CPol is more sensitive to increases of TV (Figures 4 and 5).

Increasing the mobility of the nodes has a big impact on the messages exchanged in CPol, a slight impact on CFor and it does not affect the situation detection capability of the nodes. A similar behavior is observed when increasing the spatial validity of the rules. CFor is much more robust against such changes and does not degrade the node's ASDR.

When the Lazy requesting mechanism is enabled, the results are quite analogous. However, the volume of exchanged messages is about 3-10 times lower compared to the standard scheme. We should note at this point that we also used the idea of lazy requesting for the CPol scheme in Part B experiments.

Another interesting point is that if we use the standard ASDR

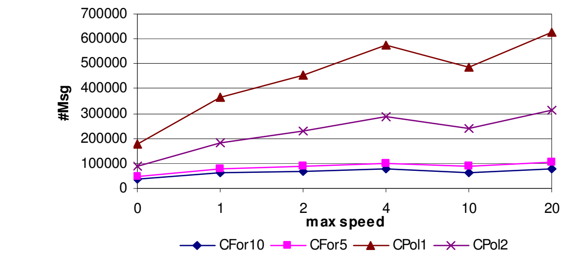

Figure 4. A1: Number of exchanged messages as a function of the maximum speed

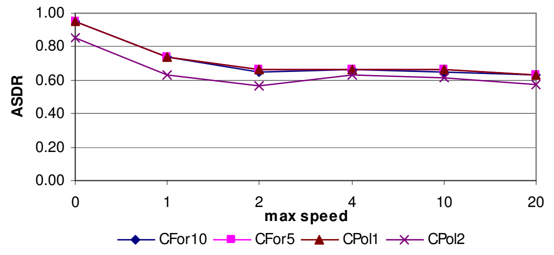

Figure 5. A1: ASDR as a function of the maximum speed

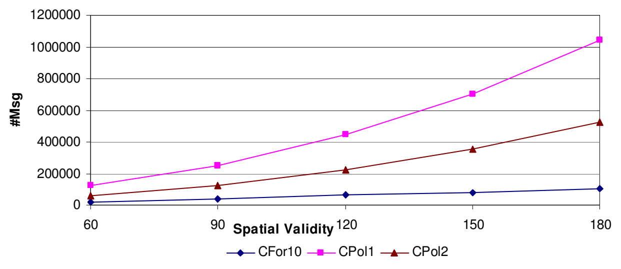

Figure 6. A2: Number of exchanged messages as a function of the spatial validity of the requests

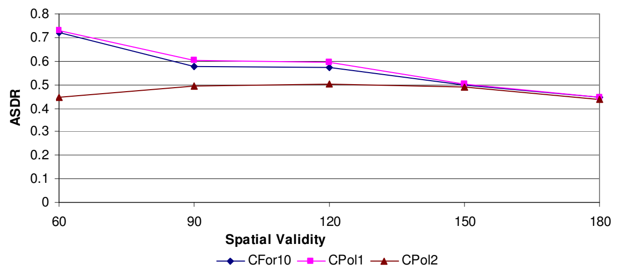

Figure 7. A2: ASDR as a function of the spatial validity of the requests

metric for Part B experiments, its values fall to ~20% (Figures 9 and 11). This is something expected, because the nodes rely more on their own sensors for detecting situation changes than on other Context Providers. Does this mean that the nodes miss some events? The answer is "No". The nodes have the same sensitivity to events but with much less message exchange and without exhaustion of their resources (due to unnecessary communications). In order to better demonstrate this we also provide in Figures 9 and 11 the results for ASDR' (the upper lines in the figures). ASDR' has the same nominator as ASDR but the

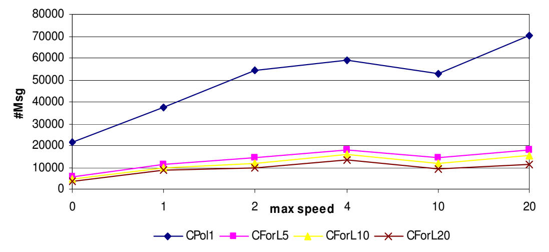

Figure 8. B1: Number of exchanged messages as a function of the maximum speed

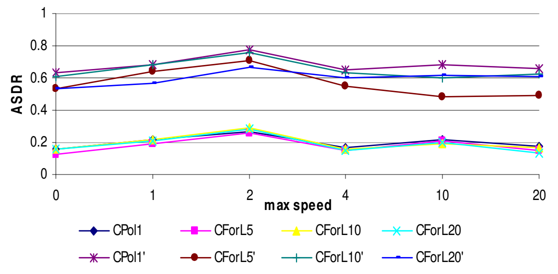

Figure 9. B1: ASDR as a function of the maximum speed

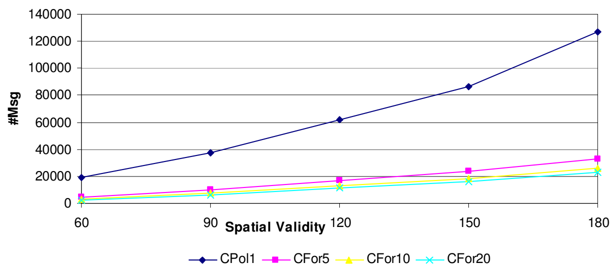

Figure 10. B2: Number of exchanged messages as a function of the spatial validity of the requests

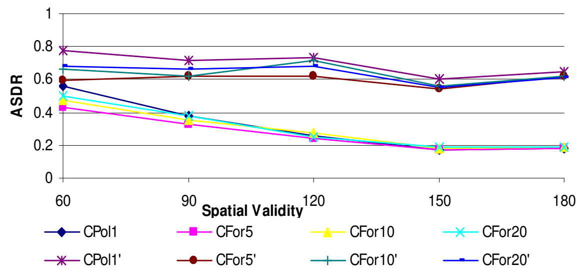

Figure 11. B2: ASDR as a function of the spatial validity of the requests

denominator takes into account only combinations where 2 out of the 3 conditions are satisfied by the requesting node itself. One can see that these values are approximately the same as those in Figures 5 and 7.

In fact, if the sensor values are not random, the ASDR' is expected to improve more, because real world situations do not tend to disappear rapidly, before even very close nodes can detect them. Hence, our next steps will be to replace the synthetic sensor datasets with real traces.

5. RELATED WORK

To our knowledge the application of the publish/subscribe paradigm to such dynamic environments, where networking protocols cannot be used, is an area with little results so far. Nevertheless, the mobile publish/subscribe model in general, has been an active research area during the last years. Several works have studied the special issues that arise when publishers or subscribers are mobile [9][10]. However, most of them assume that there is a MANET which implements some appropriate routing protocols or they employ routing-like structures for performing multicasting of the event notifications [12].

The authors in [11] do not make most of the aforementioned assumptions and in that sense their setup is similar to ours. However, their work targets other application domains where event topics are more structured and, to some degree, predefined. In our scheme, the context requests define the topics of interest and can be arbitrary, depending on the current needs of the nodes. Another difference is the way the temporal validity is defined. Context foraging assigns a validity value to requests, filters and responses, while in the approach of [11] time validity is assigned to the generated events. Moreover, it is worth mentioning that the Context Foraging scheme does not build any type of topology tables or other structures. In the light of such inherent differences, further comparison with the above schemes seems difficult, if meaningful at all.

6. CONCLUSION

In this paper we presented a new architecture for collaborative sensing and nomadic context-aware applications. The proposed scheme is based on principles of publish/subscribe systems applied on fully mobile nodes carrying sensors and/or requiring sensor values for reasoning about their context. An evaluation and comparison to an alternative scheme clearly indicated the benefits of the proposed scheme with regard to the number of messages exchanged. Specifically, in all scenarios tested, the disseminated messages are much less than those of the alternative (polling-based) scheme, with insignificant reduction in the situation detection capability of the nodes.

Despite this encouraging evaluation, several topics are still open and deserve further investigation. One of them is how can the scheme protect itself from malicious nodes that frequently register events just to flood the network. Moreover, some other features could be studied. One of them is that the time interval (time validity) between request retransmissions (Listing 1, requestContext algorithm, line 6) could depend on the node mobility or, equivalently, the neighbourhood change rate or the criticality of the rule (if it can be somehow defined). Another one is the caching of the context responses for some time units, either in the relay nodes or the CRs.

Finally, we should mention that this work is part of the IPAC (Integrated Platform for Autonomic Computing) European FP7 project [14]. IPAC aims at delivering a middleware and service creation environment for developing embedded, intelligent, collaborative, context-aware services in mobile nodes. IPAC relies on short-range communications for the ad hoc realization of dialogs among collaborating nodes.

7. ACKNOWLEDGMENTS

This work was partially supported by the European Commission through the FP7 ICT Programme in the scope of the project IPAC (Integrated Platform for Autonomic Computing), contract FP7-ICT-224395.

8. REFERENCES

[1] Kleinrock, L. 1995. Nomadic computing—an opportunity. *SIGCOMM Comput. Commun. Rev.* 25, 1 (Jan. 1995), 36-40.

[2] Balan, R. K. and Satyanarayanan, M. "The Case for Cyber Foraging", *Proc. 10th ACM SIGOPS European Workshop*, ACM Press, 2002, 87-92

[3] Mousavi, S. M. et al. "MobiSim : A Framework for Simulation of Mobility Models in Mobile Ad-Hoc Networks", *3rd IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (IEEE WiMob)*, New York, USA, 2007.

[4] Anagnostopoulos, C. and Hadjiefthymiades, S. Enhancing Situation Aware Systems through Imprecise Reasoning, *IEEE Transactions on Mobile Computing*, vol. 7, no. 10, 2008, 1153-1168

[5] Wang, X.H., et al. Ontology based context modeling and reasoning using OWL, In *Proc. of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops,* 14-17 March 2004, 18-22

[6] Chen, H. and Finin, T. An Ontology for a Context Aware Pervasive Computing Environment, *IJCAI workshop on ontologies and distributed systems*, Acapulco MX, 2003.

[7] Eugster, P.T. et al. Epidemic information dissemination in distributed systems, *Computer* , vol.37, no.5, 2004, 60-67

[8] Costa, P., et al. When cars start gossiping. In *Proceedings of the 6th Workshop on Middleware For Network Eccentric and Mobile Applications* (Glasgow, Scotland, April 01 - 01, 2008). MiNEMA '08. ACM, New York, NY, 1-4.

[9] Rezende, C. G., Rocha, B. P., and Loureiro, A. A. 2008. Publish/subscribe architecture for mobile ad hoc networks. In *Proceedings of the 2008 ACM Symposium on Applied Computing* (Fortaleza, Ceara, Brazil, March 16 - 20, 2008). SAC '08. ACM, New York, NY, 1913-1917

[10] Huang, Y. and Garcia-Molina, H. Publish/Subscribe Tree Construction in Wireless Ad-Hoc Networks. *MDM*, 2003.

[11] Baehni, S., Chhabra, C. S., Guerraoui, R. Frugal. Event Dissemination in a Mobile Environment. *ACM/IFIP/USENIX 6th International Middleware Conference,* 2005, 205-224

[12] Muthusamy, V., Petrovic, M., Gao, D., Jacobsen, H.-A., Publisher mobility in distributed publish/subscribe systems, *25th IEEE International Conference on Distributed Computing Systems Workshops, 2005.* 2005, 421-427

[13] Carzaniga, A. and Wolf, A. L. Forwarding in a content-based network. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications* (Karlsruhe, Germany, August 25 - 29, 2003). SIGCOMM '03. ACM, New York, NY, 163-174.

[14] Tsetsos et al. *D1.1 IPAC Project Presentation*, Public deliverable available at http://ipac.di.uoa.gr, June 2008