# Collaborative Sensing over Smart Sensors

Vassileios Tsetsos, Nikolaos Silvestros, and Stathes Hadjiefthymiades
b.tsetsos@di.uoa.gr, grad0887@di.uoa.gr, shadj@di.uoa.gr
Pervasive Computing Research Group,
Dept of Informatics and Telecommunications, University of Athens
Panepistimiopolis, Ilissia, 15784, Greece

**Abstract**. Sensor technology is mandatory in order to build and provide context-aware applications in modern mobile environments. However, some limitations in current solutions prevent the wide deployment and use of such applications. One of them is that all systems assume that each node has all the necessary sensors attached to it. Another one is the fact that all applications rely on proprietary sensor platforms, which restricts the deployment of these applications only to computing nodes that are compatible with these platforms. In this paper we address both issues and describe some solutions, implemented in the context of the European research project IPAC.

## 1. Introduction

Context and situation awareness are key ingredients of modern and future mobile applications. The vision is that nomadic users can freely move and still enjoy context-aware behavior from the computing systems they use. Such behavior is perceived through intelligent applications that sense the user's context and adapt accordingly. Obviously, the substrate for implementing context-aware services is to have appropriate sensing elements that can detect changes in the environment and/or user behavior. Typical sensors that are already used in many applications are weather sensors (e.g., temperature, humidity) and positioning sensors (e.g., GPS receivers). However, until now most of the proposed and existing platforms for mobile context aware applications make two basic assumptions:

a) all mobile nodes have all the sensing devices they require, and

b) the sensing component of the system has been developed on top of a specific sensor technology (i.e., a sensor solution provided by a specific vendor).

In this paper we present the approach followed in the IPAC (Integrated Platform for Autonomic Computing) project [1][7] in order to relax these assumptions and provide greater flexibility in creating context-aware systems. The first assumption is handled through a publish-subscribe scheme for exchanging sensor data between the nodes. In this way nodes that do not have attached sensors, can still exploit sensor data provided by sensor-enabled neighbors. The second assumption is relaxed by using smart sensors. Smart sensors are sensing devices that follow a specific standard and, hence, are fully interoperable. Their main characteristic is plug-and-play behavior, achieved through the Transducer Electronic DataSheets (TEDS) that are embedded in them. In the following sections we provide more details on our work.

## 2. Collaborative Sensing: the approach of Context Foraging

### 2.1 System Architecture Overview

Let us assume an architecture where several highly mobile nodes execute situation-aware applications. These are based on rules, called Situation Classification Rules (SCR), that have conditions related to context classes (e.g., Temperature, Location). An example of such rule is:

(Temperature>80) ^ (Humidity <10) ^ (Smoke=true) → Fire

The head of the rule (i.e., Fire) is the situation that holds if all conditions are satisfied. Hence, in order the nodes to demonstrate adaptive and context-aware behavior they must have the necessary context values (i.e., instances of context classes). The concept of collaborative sensing is heavily based on the assumption that not all nodes have sensors, and context values, at their disposal. In general, we have three categories of nodes in our scheme (which is called Context Foraging):

*Context Requestors (CR)*. They request context (sensor) values from their neighborhood. Each Context Request is of the form:

*CReq* := $C_i$ op $V_i$ , where $C_i$ is a context class, $V_i \in$ value set of $C_i$, op $\in$ {>, <, =, <=, >=}.

The Context Request is derived from the conditions of an SCR that cannot be locally evaluated. Each CReq has a Spatial Validity ($SV_{CReq}$) which is the range within which the context values included in the request are valid/useful for the requesting node. 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. The $SV_{CReq}$ value is "inherited" by the respective SCR that produced the CReq. This circular

region includes all nodes that can provide valid values for the *CReq* context classes. Context Requests may be disseminated periodically (we assume that an information dissemination scheme based on probabilistic broadcasting is used). This period is called the Temporal Validity of the request ($TV_{CReq}$).

*Context Providers (CP).* They transmit sensor values (Context Responses, CRes) if these match with some registered CReqs. Each context provider has an index data structure used for two purposes: a) as a registry of all context requests received, and b) as a mechanism that matches events (fresh sensor values) with CReqs. 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 requests generate events that are disseminated through the network.

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

*CRes* := $\{C_1 = V_1, \ldots, C_i = V_i\}$, $V_i \in$ value set of $C_i$.

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

The index structure used for the subscription registration and matchmaking is depicted in Fig.1. In this index, 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"). *SV* values are used for setting the spatial validity parameter in the context response. 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 (*timeout* values) that are scheduled in a typical job scheduler.

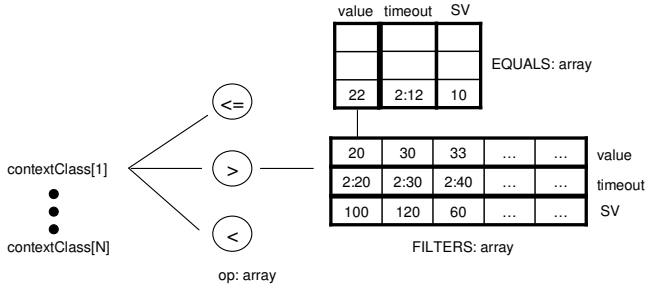


**Fig.1. The index used in Context Providers**

*Context Relays (CRel).* Nodes that do not have the sensors required by a context request or are not interested in the context response contents. CRels just forward messages.

### 2.2 Performance Evaluation

In order to assess the performance of the proposed scheme we ran several simulations, where we compared it to an alternative scheme, also suitable for nomadic computing. 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.

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 CReq, CRes and their forwards.
2) Average Situation Detection Ratio (ASDR). The average SDR over all Context Requestors. Situation Detection Ratio for a CR *i* is defined as:

$$SDR_i = A/B$$

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* is observed within the area where the rule is spatially valid.

The simulations compare the performance of the two schemes, CFor (for Context Foraging) and CPol. The setup is described in Table 1. Table 2 shows the SCR rules used in the CRs. No CR has sensors, while each CP has only one of the sensor types (context classes).

**Table 1. Simulation Setup**

| | |
|---|---|
| # of nodes | 100 |
| Mobility model | Random waypoint<br>Max pause time: 20<br>Min speed: 0 |
| # of SCR per CR | 2 |
| SV of SCRs | 110 |
| Communication range | 50 |

**Table 2. Simulation Setup**

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

Several scenarios were executed, where the parameters of Table 1 were adjusted. In all scenarios, the number of messages for CFor is much lower than for CPol, with insignificant reduction in the situation detection capability of the nodes (ASDR). Fig. 2 and 3 are two typical figures of the results. More results and a more detailed description of the Context Foraging scheme can be found in [2].

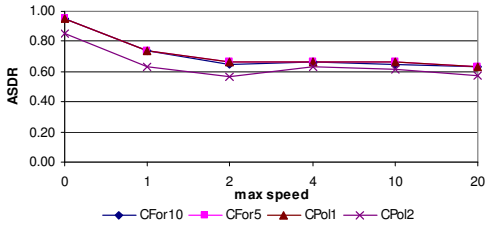


**Fig.2. ASDR as a function of the max speed**

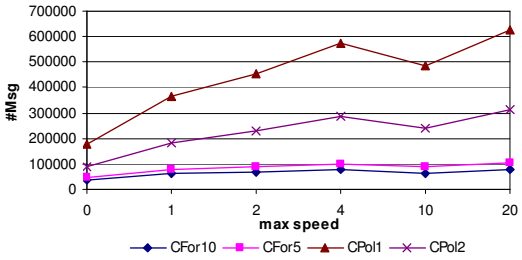


**Fig. 3. #Msg as a function of the max speed**

## 3. Smart Sensors

A "smart sensor" is a transducer (sensor or actuator) which provides extra functionality beyond that provided by a regular sensor. "Smart sensors" enable easier integration in networked environments because they process the measured data and they transform them in appropriate units. For example, a "smart sensor" that measures temperature, can transform the output voltage of a regular sensor into temperature data that represents Celsius or Fahrenheit degrees internally. This can be done because each "smart sensor" has a transducer electronic data sheet (TEDS). This is a memory device attached to the transducer, which stores some data related with identification, calibration, data correction and measurement range.

In our approach, we use "smart sensors" which comply to the IEEE 1451 family of standards [3]. IEEE 1451 describes a set of open, common and network-independent interfaces for connecting transducers to instruments, systems and networks through analog, digital, wired or wireless interfaces. Its main goals are a) the development of network and vendor independent transducer interfaces, b) to allow transducers to be hot-swapped, and c) to minimize manual system configuration.

According to these standards, each "smart sensor" consists of two main components: a transducer interface module (TIM) and a network capable application processor (NCAP). A TIM contains one or more transducers, signal processing units, A/D and/or D/A converters and an interface through which it can communicate with the NCAP. An NCAP is the system that interconnects one or more TIMs with the user network or host processor. It can have many different communication interfaces with the outside world. Through the TEDS, implemented in TIMs, one can easily have access to transducer data through network interfaces and also have the capability of "plug and play" TIMs, accessed automatically after their connection with NCAP.

The two standards of IEEE 1451 that are of interest for us are IEEE 1451.0 and IEEE 1451.2. IEEE 1451.0 describes common commands, functions and operations and the structure of the TEDS. Many different wired or wireless protocols can be used to connect TIM and NCAP. IEEE 1451.2 describes the physical layer for a point-to-point communication between an NCAP and a TIM based on serial protocols such as RS-232 and USB.

Obviously, such type of sensors are quite useful for really open and interoperable context-aware environments. However, after an extensive market survey, it turned out that no actual implementations are available, apart from some obsolete development kits. The main reason for this is the continuous

evolution of the standard that makes sensor vendors reluctant to adopt it. Hence, we are currently implementing the standard using the Sun SPOT [4] nodes as TIMs. The sensors of each TIM are the sensors that each Sun SPOT carries by default and some external ones (e.g., GPS) which are connected to SPOTs through the SPOT I/O ports. Regarding the NCAP, we develop a software version of it because, as already mentioned, no available hardware implementations are available. TIMs and NCAP communicate through a USB interface, based on the IEEE 1451.2 standard. We implement all the specifications of subsystems and interfaces that are required for building a large scale testbed for smart sensors. The communication between the NCAP and the applications is performed through an HTTP API defined in the standard. This API is invoked by an IPAC middleware service that is responsible for providing the applications access to sensors (Sensor Elements Component Proxy, SEC). The overall architecture of each IPAC node that is equipped with smart sensors is shown in Fig.4. Given this approach, an IPAC application can easily interact with different compliant sensor nodes that have the TIM-part of IEEE 1451 installed, through an abstract API.

The implementation of the serial communication (IEEE 1451.2) will be based on the open source project JDDAC [4] and particularly on Java Transducer Interface (JTI) [5]. On the other hand, no existing software implementation of IEEE 1451.0 is available and the core parts of the standard are developed from scratch.

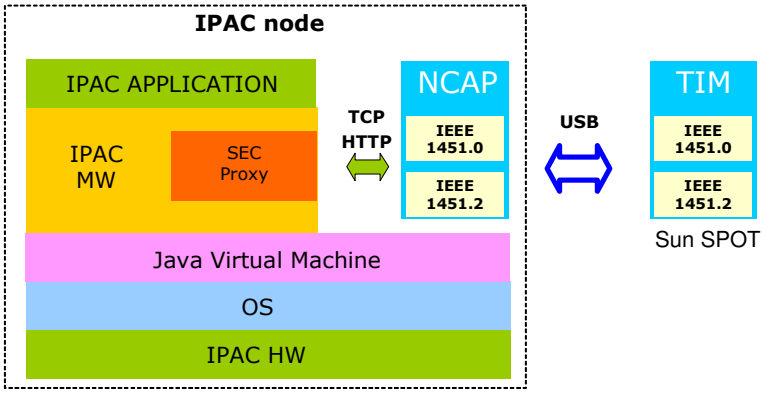


**Fig. 4. An IEEE 1451-compliant architecture based on Sun SPOTs**

## 4. Conclusions and Future Work

In this paper we briefly presented the overall architecture for a collaborative sensing infrastructure over smart sensors. The proposed approach can enable advanced context-aware applications in completely ad hoc networks of mobile nodes. Of course, in order to fully exploit such a solution, further progress is required, especially regarding the sensor infrastructure. Specifically, the IEEE 1451 standard is still evolving. Unless the standard becomes mature and stable, and standard-compliant hardware is available, the vision for such open sensing environments cannot be fully realized. We are currently finalizing the implementation of such system over the Sun SPOT platform. Our next step is to port the IEEE 1451 implementation to the Crossbow motes platform (over TinyOS) in order to test the interoperability between these two completely different platforms. Another topic that is of interest to our team is the implementation of the IEEE 1451.5 standard that refers to the wireless communication between the NCAP and the TIM. Finally, we aim at exposing our work as an open source project that can be the basis for collaboration in this very hot research area.

## 5. References

[1] Integrated Platform for Autonomic Computing, http://ipac.di.uoa.gr

[2] Tsetsos, V., Hadjiefthymiades, S. "An Innovative Architecture for Context Foraging", *8th International ACM Workshop on Data Engineering for Wireless and Mobile Access (MobIDE)*, Providence, Rhode Island, USA, June, 2009

[3] Lee, K., "IEEE 1451: A Standard in Support of Smart Transducer Networking", IEEE Instrumentation and Measurement Technology Conference Baltimore, MD USA, May 1-4, 2000

[4] Sun SPOT World, http://www.sunspotworld.com/

[5] Java Distributed Data Acquisition and Control, https://jddac.dev.java.net/

[6] Java Transducer Interface, https://jti.dev.java.net/

[7] C. Panayiotou, E. Fytros, V. Tsetsos, G. Samaras, S. Hadjiefthymiades, D. Piquet, "Integrated Platform for Autonomic Computing", *poster paper published at IEEE SECON*, Rome, Italy, June, 2009