A Reconfigurable Middleware for Context-Aware Applications in Autonomic Computing

Vassilis Papataxiarhis[*], Vassileios Tsetsos*, George Valkanas*, Corinne Kassapoglou-Faist[†], Damien Piguet[†], Stathes Hadjiefthymiades[*]

[*]*Pervasive Computing Research Group, Dept of Informatics and Telecommunications, National and Kapodistrian University of Athens, Panepistimiopolis, Ilissia, GR-15784, Athens, Greece*
Email: {vpap, b.tsetsos, gvalk, shadj}@di.uoa.gr

[†]*CSEM, Centre Suisse d'Electronique et de Microtechnique S.A. Jaquet-Droz 1, CH-2002, Neuchâtel, Switzerland*
Email: {corinne.kassapoglou-faist, damien.piguet}@csem.ch

***Abstract*—The vision of autonomic computing involves distributed nodes capable of managing and preserving themselves. In practice, autonomic computing is also strongly connected to the concepts of mobility and platform heterogeneity since next generation networks assume different types of mobile nodes that operate inside ad hoc environments. In such cases where the context changes can be frequent, the capability of capturing the environmental changes is crucial. However, it cannot be assumed that all the nodes are equipped with all possible types of sensors in order to locally retrieve the required contextual information. Sensor information exchange between the nodes through efficient data dissemination mechanisms can further improve the overall awareness of the network. Another challenging task for autonomic computing concerns the self-reconfiguration of the autonomous nodes according to the sensed context. This paper presents the overall architecture design, the implementation and the evaluation of a middleware developed in the context of the Integrated Platform for Autonomic Computing (IPAC). This middleware is targeted to embedded devices and supports mobile context-aware applications that render the aforementioned desired behavior. The paper focuses on two aspects of autonomic computing middleware: collaborative context-awareness and self-reconfiguration.**

Keywords-middleware, autonomic computing, self-adaptation, collaborative context-awareness

I. INTRODUCTION

Autonomic computing refers to a relatively new paradigm where computing devices are able to self-tune and operate in highly dynamic and mobile environments. In autonomic computing environments, there are some special requirements not applicable to typical mobile computing case. One of them is the lack of any type of structured networking. In such environments nodes do not rely on routing techniques for message exchanging. Mobility is conceived as random movement or appearance (switch-on) and disappearance (switch-off) in random places and times. Applications should be able to operate and communicate efficiently under such highly varying conditions.

Moreover, since the nodes are in a continuous change of state, context awareness should be exploited by the running applications. Towards this direction, readings obtained from sensing elements constitute data usable by applications. Such information can be used during application execution or even to support some fundamental aspects of autonomic computing paradigm such as the so called self-CHOP features (i.e., self-configuration, self-healing, self-optimization, self-protection). Reconfiguration is an important aspect of such systems since we have changing environment conditions and portable or embedded devices (e.g., netbooks, smartphones) with limited capabilities.

In this paper we present the middleware design of the IPAC platform [3]. IPAC aims at delivering a middleware and service creation environment for developing embedded, intelligent, collaborative, context-aware services in mobile nodes. IPAC supports applications that mainly exchange simple data (human-created messages, sensor values etc.) in very highly dynamic environments (e.g., vehicular ad hoc networks). The lightweight IPAC middleware stack provides all services required for the deployment and execution of diverse applications in a collaborative nomadic environment. These services are supported by novel knowledge and ontology engineering techniques, which deal with interoperability, integration, and reconfiguration problems that are met in contemporary embedded platforms. IPAC relies on short range communications (e.g., WiseMac [12]) for the ad hoc realization of dialogs between nodes and on sensing elements for detecting contextual changes and generating respective events consumed by the applications. The IPAC middleware utilizes the Open Service Gateway initiative (OSGi) framework [6] to facilitate the dynamic distribution and deployment of its service modules.

The rest of the paper is organized as follows. Section II presents the related work in the area of middleware for mobile computing. An overview of the IPAC middleware is discussed in Section III. Section IV focuses on the collaborative context awareness aspect of the platform. The knowledge based reconfiguration mechanism is described in Section V. Section VI evaluates IPAC in real-case scenarios. The paper concludes with some directions for future work.

II. PRIOR WORK

Context-aware systems and mobile computing middleware constitute major subareas where a lot of research work has been done over the past few years. SENSE (Smart Embedded Network of Sensing Entities) [7] is working on the creation of a distributed embedded network consisting of heterogeneous portable devices. The sensor infrastructure operates in a cooperative fashion to record a consolidated view. Contrary to IPAC, SENSE does not exploit high level information (e.g. application requirements) to enable reconfiguration facilities.

EMMA (Embedded Middleware in Mobility Applications) [8] aims at the deployment of a creation environment for embedded software, as well as middleware platform that will facilitate the cooperation of sensing entities in the domain of transport applications. Specifically, the project emphasizes the seamless collaboration of wireless sensing elements in order to achieve intelligent behavior of cost-effective services. EMMA services are not based on context-aware information.

DySCAS (Dynamically Self-Configuring Automotive Systems) [9] aims to develop methods, tools and architectural guidelines for self-configurable systems in the context of embedded vehicle electronics, driven by the fact that many applications could interact with mobile devices. Hence, DySCAS

considers situations such as automatic discovery and use of new devices connected to a vehicle. Contrary to IPAC, DySCAS does not investigate any algorithmic solutions in the area of information dissemination and rumor spreading.

Hydra [10] targets at networked embedded systems providing an integrated environment that facilitates the development of mobile applications. The proposed middleware supports both distributed and centralized architectures, while it does not exploit any knowledge technologies to drive any adaptation processes.

MASS (Middleware for Adaptive Semantic Support) [5] is an ontology-based middleware, aiming to enhance the development of context-aware applications. RDF(S) and OWL ontologies are used to express semantic information in mobile devices, allowing for automated reasoning and adaptation according to user profile and device capabilities. The middleware specifies the means to access semantic services, accompanied by a matching algorithm capable of identifying compatible services with user applications.

A notable difference between IPAC and the above efforts is that the majority lack in adaptation and reconfiguration of both applications and middleware. According to our knowledge, none of the existing research or industrial solutions enables a cross-layer execution and optimization of such reconfiguration capabilities in the seamless way indicated by the principles of autonomic computing. The self-reconfiguration processes performed in the IPAC nodes constitute a novel approach in the area of embedded systems since they affect several layers (applications, middleware services and hardware).

III. THE IPAC MIDDLEWARE

In the context of the IPAC platform a number of basic middleware services have been developed to support a wide range of application domains (Fig. 1). Here, we provide a brief explanation of the core IPAC modules.

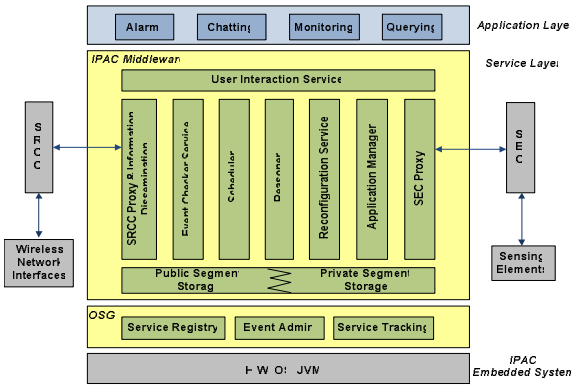

Figure 1. IPAC Middleware Architecture.

Event Checker Service (ECS). The ECS is responsible for continuously checking whether the events defined by the applications occur. An event is an application-specific Prolog-like conjunctive rule of the form:

(Sensor1 op Value1) ^ (Sensor2 op Value2) ^…^ (Sensor3 op Value3) → EventName

where *op* is some comparison operator. The real innovation in the functionality of ECS is that it implements the concept of collaborative sensing. If a node is not equipped with the sensors required to detect the application events, a mechanism for the exchange of sensor values is exploited. We call such functionality "*context foraging*" (Section IV).

Reconfiguration Service. The Reconfiguration service takes into account the node's contextual information and adapts the node behavior and settings accordingly. This adaptation process affects parameters associated with the functionality of middleware services (e.g., the transmission rate of information dissemination algorithms, the device configuration, the user interaction), network (e.g., all nodes switch to the same communication interface) and, indirectly, the applications.

Reasoner. This service constitutes the core component for realizing the autonomic behavior of the IPAC nodes since it is responsible for performing knowledge-based inferences. Reasoner drives the self-adaptation process of the IPAC nodes and checks possible conflicts regarding the resources shared among the IPAC applications. IPAC middleware takes advantage of the J2ME (MIDP 2.0, CLDC 1.0) version of Java Internet Prolog (JIProlog) engine [1], which is a cross platform framework that enables the development of lightweight reasoning services on devices with restricted resources. This solution works better than reasoning over other knowledge representation methodologies that have been investigated, such as lightweight ontologies, since no efficient reasoning modules are available for such formalisms in resource constrained devices [11].

Storage Service. The storage layer consists of the private and the public segments. The private segment is used by applications and modules that run on the node and need data storage functions. The public segment is used for storing measurements and messages that may be forwarded to other nodes in proximity.

SRCC Proxy and Information Dissemination Service. IPAC nodes exchange messages using the Short Range Communication Component (SRCC) Proxy and the Information Dissemination service. The SRCC Proxy is an abstraction of the SRC hardware, seamlessly providing basic networking functionality (e.g., activation of wireless interfaces, transmission and reception of messages) regardless of the underlying protocol. The Information Dissemination Service lies on top of it and implements the adopted dissemination algorithm [2].

SEC Proxy. The Sensing Elements Component (SEC) Proxy is responsible to manage the sensing devices hosted on the SEC and make possible their interfacing with the middleware services in a uniform way. The SEC Proxy performs the following tasks:

1. Discover sensors recently connected onto the node.
2. Configuration of sensors (H/W settings, sampling rate).
3. Data collection from the local sensor elements available.
4. Storage of the acquired sensor data.

Our approach is based on "smart sensors", which comply with the IEEE 1451 family of standards [13]. IEEE 1451 describes a set of open and network-independent interfaces for connecting transducers to instruments, systems and networks. 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. 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 D/A converters and an interface through which it can communicate with the NCAP. NCAP is the system that interconnects the TIMs with the user network or host processor. We have implemented the standard using the Sun SPOT [14] nodes as TIMs. The sensors of each TIM are the sensors that each Sun SPOT carries by default along with some external ones (e.g., GPS) which are connected to SPOTs through their I/O interfaces. Regarding the NCAP, we proceeded with a software implementation since no hardware implementations are available up to now. TIMs and NCAP communicate through a USB interface while the communication between the NCAP and the applications is performed through an HTTP API (invoked by the SEC Proxy). Fig. 2 shows the overall architecture of an IPAC node equipped with smart sensors.

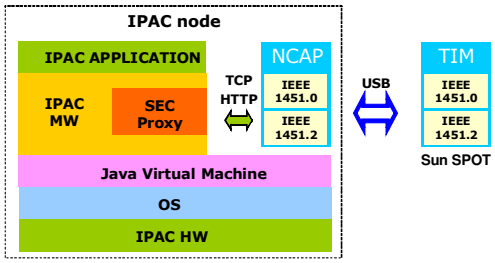

Figure 2. An IEEE 1451-compliant architecture based on Sun SPOTs.

IV. COLLABORATIVE CONTEXT-AWARENESS

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 a rule is:

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

The head of the rule (i.e., Fire) is the situation that holds true if all conditions are satisfied. In order for the nodes to demonstrate adaptive and context-aware behavior they must have the necessary contextual 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 which corresponds to a realistic scenario. Hence, in IPAC we have adopted a collaborative scheme for context-awareness, termed Context Foraging (CFor) [4]. In CFor, three types of nodes can be distinguished:

Context Requestors (CR). They request context (sensor) values from their neighborhood. The Context Request (CReq) is derived from the conditions of an SCR that cannot be locally evaluated. Each request has a Spatial Validity (SVCReq) which is the range within which the context values included in the CReq are valid. The requests also have a Temporal Validity (TVCReq) that is the period with which the CReqs are disseminated according to the adopted probabilistic broadcast scheme.

Context Providers (CP). They transmit sensor values (Context Responses, CRes) if these match with some registered CReqs. A context response has also a spatial validity parameter which is the maximum of the individual spatial validity values included in the response. 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.

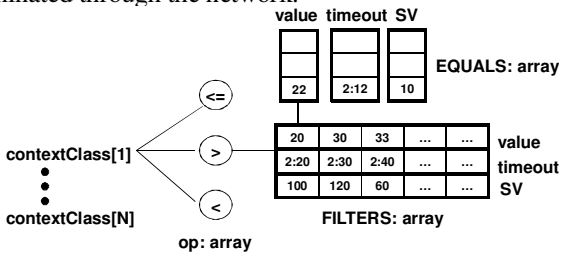

Figure 3. The index used in Context Providers.

The index structure used for the subscription registration and matchmaking is depicted in Fig. 3. The FILTERS array contains the context conditions 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 constitutes an optimization in order to avoid unnecessary filters (e.g., filters that overlap with the existing ones). SV values denote the spatial validity parameter of the context responses. Timeout values are also used to remove filters that their validity has been expired.

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.

The proposed scheme has been evaluated through several simulations, where we compared it to Context Polling (CPol), which is a suitable scheme for nomadic computing. IPAC seems to behave better in all kind of simulated scenarios. Specifically, the number of messages for CFor is much lower than for CPol, with insignificant reduction in the situation detection capability of the nodes. A detailed description of the Context Foraging scheme and the simulated scenarios can be found in [4].

V. KNOWLEDGE-BASED RECONFIGURATION

A. IPAC Models

Sensor model provides a common vocabulary about sensors and their features. It defines the basic characteristics of sensors and the available sensor types (e.g. GPS receiver, smoke detector). Such type of information is modeled through predicate hierarchies (taxonomies) in order to take advantage of instances classification during the execution of reasoning processes.

Node profile defines concepts and relationships that refer to the basic features of an IPAC node. Some of the metadata belonging to the node profile are the available communication interfaces, the available storage space and the supported UIs.

Application profiles. Each application has its own profile that describes its features, the preconditions for being deployed and the events that the application is interested in. An application profile example in terms of Prolog is provided below:

usesInfoFromSensor(appID03, smoke_sensor).
usesInfoFromSensor(appID03, temp_sensor).
requiresUI(appID03, visual).
event(fire_alarm) :- smoke_sensor>=0.7, temp_sensor>=20.

In this example, in case of smoke detection (probability>=0.7) and of temperature over a limit value (20 degrees), a "fire_alarm" event is raised (*application policy*).

Reconfiguration policies. Besides the reconfiguration requests that stem from the applications, IPAC middleware also supports the execution of policies that enforce updates in its settings in order to achieve optimal operation of the node. These dynamically updated policies aim to prevent the appearance of unacceptable situations that could aggravate the system functionality or in case the system status is error-prone. Similarly to the application policies, the reconfiguration policies are represented in a declarative manner. An example follows:

policy(hasCommInterface,X,ieee_802_11):-
numberOfNeighbors(X,N), N=0, node(X),
hasCommInterface(X,wisemac).

The above policy states that in case there is no other node in the neighborhood, the communication interface should change to reach nodes through some other technology.

B. Reconfiguration Functionality

In IPAC, the middleware services that are responsible for altering the operation/configuration of the node are called through a central dispatching mechanism, the Reconfiguration Service. The rationale behind this orientation is that the service and device settings are shared resources and should be controlled

by a central entity. Moreover, global and cross-layer knowledge may be necessary for some reconfigurations. Based on these principles, the reconfiguration workflow is as follows: an application sends a request for reconfiguration (it may affect a service, the device settings or the network). There are two main types of requests: the soft and the hard ones. The former are associated with some "timeValidity" parameter (e.g., "whenever, within the next five minutes, the UI is able to switch to sound mode, do it"). The applications do not require feedback from the system and should not make any assumptions on such requests. At the most, they can receive a notification when the requested change is performed. On the other hand, the hard requests should be executed immediately (if at all) and the application should receive some feedback whether the reconfiguration has been performed or not. Once a reconfiguration request arrives in the Reconfiguration service, the Reasoner is invoked and decides if it is consistent with the current system status. If not and it is

- a soft request with "timeValidity" set, then it reschedules the request and checks it after a specified period of time.
- a hard request, it sends a response "Reconfiguration not possible" to the Response Queue.

If the modification is possible, a message with the requested parameters is sent to the Dispatcher Queue. The Reconfiguration service consumes this message and performs the adaptation. The result is inserted to the Response Queue ("Success" or "Failure"). Finally, once a message destined for an application arrives at the Response Queue, the respective application consumes it and continues executing its application logic.

An example that describes the main processes of the reconfiguration phase follows. This scenario assumes that an IPAC node (e.g., with id *node03*) supports both visual and audio UIs. Initially, it is assumed that the visual interface has been activated. Such knowledge is explicitly captured in the node profile, through the following Prolog statements:

supportsUI(node03, visual).
supportsUI(node03, audio).
hasUI(node03, visual).

The scenario considers that application A (e.g., with id *app02*) is stored on an IPAC node in order to be deployed while other applications are already running and A requires audio interface to run properly (e.g., the statement *requiresUI(app02,audio)* is part of its profile). First, the application manager checks whether the requirements of the new application are consistent with those imposed by the applications that already run. This is achieved through the Reasoner that performs a consistency check with the profiles of all currently running applications. Assuming that there is no inconsistency, the UI can be switched to audio without any conflicts and a reconfiguration request is scheduled to perform the desired action. Similar actions are performed for switching on and off the communication interfaces and for enabling/disabling sensors.

VI. IPAC In Humanitarian Relief Operations

A. Scenario Description

This section presents a use-case trial that demonstrates the capabilities of the platform and helps to evaluate the requirements, and thus backing up the motivation of this work. The scenario focuses on the deployment of IPAC in simulated humanitarian operations in order to provide a much needed communication infrastructure for the execution of multiple applications in a collaborative nomadic environment. In this case, the IPAC infrastructure assumes the role of communication support between pedestrians, and vehicles, or between static check points and vehicles or pedestrians of the operations workforce. The demonstration took place in a wide open area at the Centro Sicurezza circuit in Turin. All experiments were performed in ASUS EeePC 900 netbooks with an Intel (R) Celeron (R) processor running at 900MHz and 1GB of main memory. These nodes used both IEEE 802.11 and WiseMac communication interfaces and were equipped with a number of sensor devices to capture context information.

The scenario concerns the detection of bad weather conditions while a number of vehicles are moving arbitrarily inside an area. The weather is characterized by low temperature and snowfalls and the possibility of ice presence on the road. To identify hazardous and unsafe conditions in time, IPAC adopted the *context foraging* approach (Section IV). Specifically, the scenario assumes three vehicles that are moving at the same direction. One of the car nodes is equipped with a camera capable of detecting road curvature and ice/snow (the vision sensor) while the remaining nodes are equipped with temperature sensors. There is also one fixed node along the roadside to relay information of high priority. The rationale behind this node is to act as a "beacon" in case of receiving possible alerts or warnings.

The scenario demonstrates a network reconfiguration process for power saving. Specifically, at the time a node needs to broadcast longer messages such as sensor measurements, it has to take advantage of the IEEE 802.11 interface that allows for transmitting large packets. Hence, it first broadcasts a short "LEAVE-LOW-POWER" message to the network in order to enable the activation of the WiFi interface in all the car nodes. This way, the nodes are capable of transmitting and receiving sensor measurements. The "LEAVE-LOW-POWER" message comes with a time validity parameter that is taken as the time-to-leave (TTL) value of the corresponding IPAC header, which denotes the time period that the WiFi interface should be active. When this time validity parameter expires, the network nodes switch back to low power state (turn the IEEE 802.11 off).

B. Reconfiguration Performance

The reconfiguration process was evaluated though the following parameters have been quantified in the above scenario:

(a) the response time of the reconfiguration service and
(b) the total time required to perform reconfiguration.

The first measurement concerns the sensitivity of the system according to context changes. A modification of the environment (e.g., absence of other nodes in the neighborhood) may lead to the execution of certain reconfiguration policies (e.g., switch to another communication interface). Fig. 4 presents the times between a context change that took place and the corresponding reconfiguration. Specifically, these are the mean times needed to perform the node adaptation across several policy checking periods indicating the degree in which the system is responsive to context changes. The presented times include (a) the update of the knowledge base according to the modification, (b) the policy execution, (c) the check of the applicability of the requested change, and, (d) the execution of the reconfiguration. Since the policy execution is performed periodically, the required time is mostly dependent on the policy checking period. One can observe that the sum of the above times is approximately the half of the period used. For example, using a policy checking period of 1sec, the mean reconfiguration time is 0,54s, while setting the period to 20s the mean reconfiguration time is 10,425s. This is anticipated since context changes occur at random. Hence, taking into account that checking the policies does not require a significant amount of time, the period can be set to a small value (e.g., 1s) in order to increase the system throughput.

Regarding the network reconfiguration experiments, the total time between the initial request and the actual reconfiguration of the node settings is measured. The clocks of all the nodes participating to the scenarios had to be synchronized. The experiments demonstrated that these times mainly depend on the network traffic (i.e., the time needed for the delivery of the "reconfiguration" request across the network) and the time needed inside the middleware services to "encode"/"decode" the request. In our scenario the total network reconfiguration time takes about 7,5s and includes the following tasks:

- message delivery by the ECS to the SRCC
- dissemination of the message to the network
- message reception by the SRCC of the second node
- message delivery by the SRCC to the Storage service and event-based communication with Reconfiguration service
- requested modification takes place (if feasible).

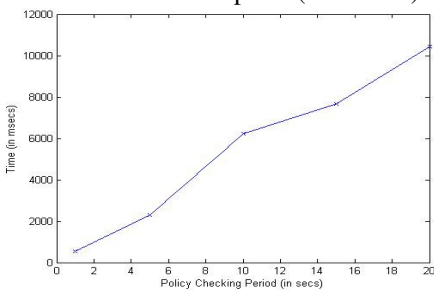

Figure 4. Reconfiguration time w.r.t. policy checking period

C. *Network Analytics*

In the humanitarian scenario, the IPAC network has 4 nodes. This paragraph performs an analysis on the way the WiseMac-based network performed, trying to assess its responsiveness and detect any possible congestion or message loss. It is based on message logs collected using a WiseMac sniffer (independent, passive node). In networks with few nodes we expect a high retransmission rate in the probabilistic broadcast algorithm (retransmission probability equal to 0.9 for 4 neighboring nodes). Retransmissions are controlled on one hand by the retransmission attempt periods as a function of criticality adopted for the scenario, the criticality of the messages set by the message sender entity and the neighbor lifetime. The retransmission attempt periods were set to 2s, 5s and 10s respectively for high, medium and low criticality. The neighbor lifetime was set to 10s to accommodate the application periodicity and mobility.

The scenario starts with subscription of messages (remote triggers). The node having the vision sensor registers to temperature measurements, while the other car nodes register to ice alerts. These messages have a "HIGH" criticality and a very long TTL, meaning that they are repeated every 5s throughout the scenario execution. To demonstrate the interface re-configurability feature, the node having a temperature sensor emits "LEAVE-LOW-POWER" instructions in order to send the temperature data through the WiFi interface. The third type of packets in this scenario contains the temperature measurements (longer packets). Finally, the ice alert message is a very short packet. All messages have "HIGH" criticality, causing frequent repetition attempts. Sniffer logs show an exaggerated importance of the "LEAVE-LOW-POWER" messages with respect to the ice alert and temperature value messages. Having few nodes, message repetitions account for 72% of the circulating packet.

VII. Conclusions And Future Work

In this paper we have discussed the IPAC approach for a middleware able to support context-aware applications for autonomic computing environments. IPAC enables several desired features of such applications, such as self-reconfiguration and collaborative applications. The paper presents the overall middleware architecture describing the design approaches followed for the development of the platform and focuses on two basic aspects: reconfiguration capabilities, and collaborative context-awareness. The scalability of the platform under realistic conditions according to an extensive real-case scenario was shown. Future work includes the improvement of the policy checking mechanism of IPAC nodes by optimizing the period parameter and by adopting event-based mechanisms that identify policy violations (e.g., through database triggers). Finally, we are planning 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.

Acknowledgment

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.

References

[1] JIProlog - Java Internet Prolog, http://www.ugosweb.com/jiprolog/index.aspx

[2] Sekkas, O., Piguet, D., Anagnostopoulos, C., Kotsakos, D., Alyfantis, G., Kassapoglou-Faist, C., Hadjiethymiades, S.: Probabilistic Information Dissemination for MANETs: the IPAC Approach. 20th Tyrrhenian International Workshop on Digital Communications, Pula, Italy (2009)

[3] Panayiotou, C., Fytros, E., Tsetsos, V., Samaras, G., Hadjiefthymiades, S., Piquet, D.: Integrated Platform for Autonomic Computing. Poster paper, IEEE SECON, Rome, Italy (2009)

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

[5] Corradi, A., Montanari, R., Toninelli, A.: Adaptive Semantic Middleware for Mobile Environments. Journal of Networks, Academy Publisher, Vol.2 Issue 1 (2007)

[6] OSGi Alliance. About the OSGi service platform. Technical Whitepaper Available at http://www.osgi.org. (2002)

[7] Bruckner, D., Kasbi, J., Velik, R., Herzner, W.: High-level Hierarchical Semantic Processing Framework for Smart Sensor Networks. In: IEEE Human System Interaction Conference 2008, Krakow (2008)

[8] Katramados, I., Barlow, A., Selvarajah, K., Shooter, C., Tully, A., Blythe, P.T.: Heterogeneous sensor integration for intelligent transport systems. Road Transport Information and Control, 2008, Manchester, UK (2008)

[9] Anthony, R.: Policy-Based Autonomic Computing with integral support for Self-Stabilisation. In: International Journal of Autonomic Computing (IJAC), ISSN (Online): 1741-8577, ISSN (Print): 1741-8569 (2009)

[10] Eisenhauer, M., Rosengren, P., Antolin, P.: HYDRA: A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems, In: 20th Tyrrhenian Workshop on Digital Communications. Pula, Sardinia, Italy (2009)

[11] Papataxiarhis, V., Tsetsos, V., Karali, I., Stamatopoulos, P., Hadjiefthymiades, S.: Developing rule-based applications for the Web: Methodologies and Tools. In: Giurca, A., Gasevic, D., Taveter, K. (eds.) Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches. Information Science Reference (2009)

[12] El-Hoiydi, A., Decotignie, J.D.: Wisemac: An ultra low power mac protocol for the downlink of infrastructure wireless sensor networks. In: ISCC2004, the 9th IEEE International Symposium on Computers and Communications, Alexandria, Egypt (2004)

[13] Lee, K.: IEEE 1451: A Standard in Support of Smart Transducer Networking. In: IEEE Instrumentation and Measurement Technology Conference Baltimore, MD USA (2000)

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