A Knowledge Plane for Context-Aware Applications in Autonomic Computing

Vassilis Papataxiarhis
National and Kapodistrian University of Athens, Panepistimiopolis, Ilissia, GR-15784, Athens, Greece
vpap@di.uoa.gr

Vassileios Tsetsos
National and Kapodistrian University of Athens, Panepistimiopolis, Ilissia, GR-15784, Athens, Greece
b.tsetsos@di.uoa.gr

Stathes Hadjiefthymiades
National and Kapodistrian University of Athens, Panepistimiopolis, Ilissia, GR-15784, Athens, Greece
shadj@di.uoa.gr

ABSTRACT
Autonomous mobile nodes constitute a vision on the computer science for over a decade. The key feature of such nodes is their ability to dynamically adapt to contextual changes, through proper reconfiguration mechanisms. In this paper we present a framework for such self-adaptation that is mainly based on knowledge management technologies. Nodes specify their reconfiguration policies through rules and their reasoning processes are responsible for enforcing them. The approach adopted is cross-layer and not restricted to specific reconfiguration scenarios. The architecture of this framework and its applicability to modern mobile networks is clearly described. Moreover, an experimental evaluation has been performed for both real and simulated scenarios.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures - Domain-specific architectures

General Terms
Design, Management, Performance

Keywords
Autonomic computing, context-aware applications, self-adaptation

1. INTRODUCTION
Mobile ad hoc networks have found many real-life applications in the last years. Their advantage of being independent of any communication infrastructure has rendered them a perfect solution for many application domains, such as crisis management, first response, and vehicle-to-vehicle services. However, besides the ad hoc communications, what would be more interesting is their potential capability to behave in an autonomous way in all layers of operation. Such concept of autonomic behavior is referenced very often in the literature [19] and is also closely connected to reconfiguration [14]. Several approaches have been also proposed for achieving real autonomic behavior in mobile networks (either structured or not) [15] [16]. However, most of them suffer from three basic limitations:

- They focus on very specific reconfiguration actions (e.g., adjustment of some communication protocol parameters).
- They do not address cross-layer reconfiguration in a realistic way. By "cross-layer" we mean that changes on the status of some layer (e.g., application) impose changes to certain functions of some other layer (e.g., network).
- The implementation of the reconfiguration policies is not very easy and lacks extensibility. In most cases, the reconfiguration rules are hard-coded in the algorithms (e.g., routing [17]) and new reconfiguration policies are hard to apply.

In this paper we try to address all these issues in a realistic way. Hence, the contributions of the proposed reconfiguration framework can be summarized to the following:

- Reconfiguration policies can be defined for various elements/operations of a node (e.g., operation of peripheral devices, networking, application management). Hence, a very broad range of reconfiguration actions, at all layers, can be supported.
- Such policies are defined through declarative rule languages and exploit knowledge models describing the node status and characteristics. Writing and updating such policies is a fairly easy process and does not require changes in the algorithms implementing the various services in the node.

The main motivation for this work was provided by the EU-funded IPAC (Integrated Platform for Autonomic Computing) research project (ICT framework). In Section 2, we describe the basic ideas and ingredients of the IPAC platform. Next, we describe in more detail the architecture and implementation of an IPAC node in Section 3. In the same section the middleware services implemented in the context of the IPAC platform are presented. The basic mechanisms for reconfiguration and the relevant workflows are discussed in Section 4. In this section we also present a sample use case for better describing the reconfiguration internals. The interested reader can find more details on the implementation of the proposed knowledge-based framework in Section 5. In Section 6, an extensive experimental

ICPS2010, July 13-15, 2010, Berlin, Germany.

evaluation is performed that demonstrates the functionality of our system. The experimental results involve the deployment of the proposed system in real mobile devices as well as simulated scenarios. Finally, the paper concludes with some related work (Section 7) and directions for future research (Section 8).

2. MOTIVATION

2.1. The IPAC Platform

This work has been performed in the context of the Integrated Platform for Autonomic Computing (IPAC) project [11]. IPAC aims at the delivery of a service creation and runtime (service provision) environment for autonomic nodes. IPAC tries to address several challenges of autonomic computing, such as reliable and efficient algorithms for information dissemination in autonomic environments, developer-friendly application creation, automatic discovery of deployed sensors and knowledge-based node reconfiguration. In this subsection, we briefly describe the overall platform. In the following sections we will focus on the reconfiguration facilities of the platform.

The main parts of the IPAC platform (Figure 1) can be summarized as follows:

- A developer-friendly graphical user interface (GUI) for building and debugging IPAC applications [18]. This GUI also comes with a domain-specific application definition language and the respective workflow language that enable developers to write applications in an intuitive way.
- Short range communication (SRC) technologies along with a novel probabilistic information dissemination model. This model is based on the concepts of [2] and is appropriate for environments with nomadic nodes having limited energy resources.
- Knowledge-based reconfiguration for embedded systems. IPAC is one of the first attempts to support mobile reasoning and related mechanisms for providing mobile service intelligence. Similar work in this area can be found in [3] [4].
- Adoption of the IEEE 1451 [10] standard for implementing plug-and-play (smart) sensors. IEEE 1451 is an evolving standard that promises a new era of sensor-enabled applications, through easy integration of diverse sensor technologies.
- Collaborative context-awareness. IPAC nodes can use contextual information for adapting the application execution even if they do not have attached sensors. Specifically, they can "harvest" sensor data and contextual events from their neighbourhood through a publish/subscribe mechanism [5].

The IPAC platform aims at supporting embedded, intelligent, collaborative and context-aware applications in mobile nodes. IPAC can support a wide variety of applications targeting to a large group of users, in diverse environments and application domains. It is a flexible platform capable of implementing quite diverse application scenarios. It may be used in simple messaging systems, such as advertisements or weather updates, in emergency updates or even in a closed group of members where confidentiality is a prerequisite. In general, 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). Of course more sophisticated applications can be supported but the main intention is to provide simple applications that can be created and used by a large target group of users, in diverse environments and application domains. The IPAC middleware provides all the required basic functionality, in the form of services, to the deployed applications.

2.2. Requirements for Reconfiguration

Mobile and autonomic computing environments contain the concept of dynamic changes and updates by nature. Hence, in order to effectively support context-aware applications in such environments, adaptivity to context changes is a crucial issue. Reconfiguration, at least in the context of this paper, involves all node settings that affect resources used by the deployed applications. Examples of such resources are: the communication interface and protocol, the user interface modality and layout, the storage allocated to each application and the application lifecycles.

In the context of the IPAC platform, three essential types of reconfiguration actions have been identified and supported:

(a) Periodic checks for reconfiguration. This type of reconfiguration is triggered by the middleware itself and not by some application request. It mainly tries to "optimize" the node operation. Such optimization may refer to various system parameters, such as energy consumption, and execution of applications that cannot be executed with the current configuration.

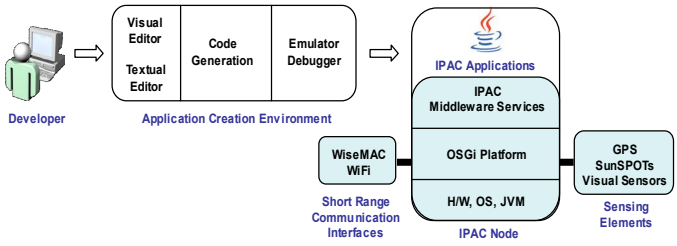

Figure 1. The IPAC Ecosystem

(b) Check for reconfiguration on application startup. An application may not be able to run, even if it can be supported in general by the node. This may happen if the application needs some resources that are currently used by other applications, or the current settings (as defined by already running applications) do not match the preconditions for execution of the blocked application.

(c) Explicit request by application. Every application is able to request certain reconfiguration actions to be performed during its execution. The middleware checks if such operations can be applied and informs the corresponding application respectively.

3. SYSTEM OVERVIEW

3.1. Node Description

The architecture of an IPAC node is depicted in Figure 2. The main components of a typical IPAC node follow:

IPAC middleware. It provides all the services required for supporting context-aware applications in nomadic environments. More details on the way context-awareness is supported in IPAC can be found in [5].

The hardware, OS and Java Virtual Machine (JVM) of the node.

OSGi services [9] running within such a framework and providing utilities to the IPAC middleware services.

Short Range Communication Component (SRCC). This component encompasses all the wireless communication capabilities required for the ad hoc networking of the node. Specifically, it provides various short range communication interfaces (such as IEEE 802.11 ad hoc mode, WiseMac [12]). The communication is based on a probabilistic broadcasting data dissemination algorithm [8] that suits such dynamic mobile environments. The information dissemination schemes are also implemented here.

Sensor Elements Component (SEC). It provides retrieval of measurements from sensors attached to the node and advanced sensor management routines (e.g., automatic discovery of new sensors, change of sampling frequency).

IPAC applications. They are self-described and self-contained applications that include the application logic, an application profile, and, optionally, some user interface.

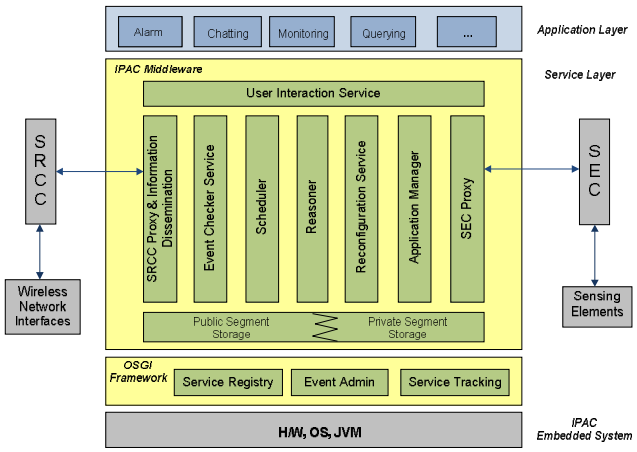

Figure 2. IPAC Node Overview

3.2. Middleware Components and Services

Inside the middleware there are several services and components. In this section we will describe how these components are involved in the reconfiguration processes:

Application Manager. It controls and manages the application lifecycle on the node and makes the application profile available during the reconfiguration processes.

Scheduler. This service is used by all other middleware services that need to schedule tasks that are performed periodically or in predefined moments in the future. The reconfiguration processes exploit scheduling for periodically checking and enforcing the corresponding policies.

SRCC Proxy. This component is responsible for abstracting the Short Range Communication interfaces towards the applications. These interfaces are regarded as shared resources and are subject to reconfiguration.

SEC Proxy. Similarly to the SRCC Proxy, this component abstracts the sensors attached to a node.

Storage Service. This component makes information available for later use, either within the node or through transmitting it over the radio interface.

Reasoner Service. The Reasoner service constitutes the core component for realizing the autonomic behaviour of the IPAC nodes. Specifically, it is a Prolog-based engine able to support inferences over the IPAC models and policies. Reasoner is able to drive the self-adaptation process of the IPAC nodes and check possible conflicts regarding the resources shared among the IPAC applications, as well.

Reconfiguration Service. The reconfiguration service manages all the processes that affect the self-adaptation of the node and the configuration of its settings. This service uses the API provided by the Reasoner service in order to perform certain reasoning tasks. More details are provided in the rest of the present paper.

Event Checker Service (ECS). It implements the event-driven paradigm of the IPAC approach. For example, it is responsible for checking the conditions that may trigger the events defined in the profile of an application and may lead to a reconfiguration action.

User Interaction Service. An application may need to receive input or give feedback to a human user. Similarly to the SRCC Proxy, the user interfaces provided by this component are considered as subject to reconfiguration.

4. IPAC KNOWLEDGE PLANE

Both context-aware re-configuration and interoperability between nodes with different features call for an intelligent system behavior, depending on the characteristics of each individual case (contextual information, node features, number of adjacent nodes, etc.). To meet these requirements, an architecture shift is necessary in the design of embedded systems middleware. Specifically, a new approach similar to the Knowledge Plane [23] is called for. Such approach includes all the necessary components in order to create a distributed cognitive system, which is aware of its goals, limitations, and resources. The IPAC Knowledge Plane operates also as a broker since it is able to disseminate to all layers of an IPAC node the status of the SRCC, SEC and other key modules of the service layer. For example, it may give feedback to the dissemination algorithms about the physical-layer networking operation of peer nodes. Some of the functions of this knowledge-based framework follow:

- models the possible situations (i.e., context) of the node/system,
- stores the situation-information collected from the information sources (e.g., sensors),
- reasons over contextual data,
- identifies possible conflicts in the system configuration,
- infers information based on real-time observations.

The ingredients of this plane are described in this section.

4.1. Language Expressiveness and Tractability of Reasoning

The selection of the appropriate knowledge technologies involved two main steps: (a) the selection of an expressive logical formalism that meets the specified requirements (e.g., definition of rules and policies), and, (b) the selection of an efficient module able to reason over such language. The nature of the IPAC platform imposes certain restrictions regarding the adopted knowledge technologies. Since IPAC targets at mobile devices, the knowledge-based components of the middleware architecture had to be implemented with lightweight technologies that offer: (a) tractable reasoning services (i.e., low reasoning times and memory requirements), and, (b) compact representation of the knowledge bases.

Regarding the selection of the appropriate knowledge representation language, the desired expressiveness is that of typical (Horn) rules with conjunctions of predicates in the body and single predicates in the head of the rules. Prolog-based implementations provide a very mature technology and its syntax and representation is very compact, in comparison to other technologies. Before deciding on the reasoning module that was used in IPAC, other solutions were also investigated, such as forward-chaining rule engines, and Description Logic reasoners. However, such solutions do not provide efficient reasoning services, thus making the execution of even simple reasoning tasks on embedded devices hard. More details about the adopted formalisms and tools can be found in Section 5.

4.2. Models and Profiles

In the context of this work, specific models are exploited in order to provide a common vocabulary to both the middleware services and applications. These models target at facilitating the self-reconfiguration of the nodes. Since they are used by the Reasoner and the Reconfiguration services of the IPAC middleware, the models are expressed in a declarative way through Prolog statements. Specifically, the following models and profiles are considered:

4.2.1. Application Profile

The application profile mainly consists of simple expressions representing the basic features of the IPAC application. These expressions concern generic description of the application or the requirements that a node should satisfy in order to deploy and execute the application. The former refers to information such as the application name, the application ID, its version or the supported user groups. Such knowledge is used in order to identify whether an application should be deployed to a node (e.g., newer version) or to allow user to join a group in the context of this application. As a result, the application profile takes advantage of the vocabulary provided by the sensor model.

On the other hand, the execution requirements of an IPAC application typically concern preconditions that a node should fulfill before running the application such as communication requirements (e.g., communication range required, average size of message payload) and application parameters (e.g., required types of user interfaces, estimated storage space). These requirements are matched with the node capabilities provided by the node profile within the IPAC middleware. If such matching is successful (i.e., the node satisfies the application requirements) the application is deployed to the node and its execution is started. Otherwise, the node either deletes it or deploys it and sets it to an inactive state.

Furthermore, the application events that constitute part of the application workflow are expressed declaratively in the application profile. Every application registers the types of information that is interested in through these events. The respective services (e.g., ECS) are responsible for creating the events at runtime by checking sensor data, incoming messages, etc. Each event is defined through a name and a number of conditions. A simple example demonstrating an application profile is provided below:

```
usesSensor(appID03, smoke_det_1).
type(smoke_det_1, smoke_detector).
usesSensor(appID03, temp_sensor_1).
type(temp_sensor_1, temperature_sensor).
requiresUI(appID03, visual).

event(fire_alarm) :- smoke_det_1>=0,
temp_sensor_1>=20.
```

In this example, the profile of an application that uses a smoke detector and a temperature sensor and requires a visual interface is given. In case of smoke detection and temperature exceeding a limit value (i.e., 20 degrees), a specific event named “fire_alarm” is raised.

4.2.2. Sensor Model

It provides a common vocabulary about sensors and their features. This way, a set of common terms is shared between the nodes and the different components of the platform, as well. This model defines the concepts (i.e., classes, terms) that concern the basic characteristics of sensors and the relationships among them. Specifically, it contains information about the type of the sensor (e.g. positioning sensor, movement detector). Such information is modelled through predicate hierarchies (taxonomies) in order to take advantage of instances classification during the execution of reasoning processes. For example, a sensing element that is defined as a temperature sensor is also classified as a sensor (which could be considered as the top class) aiming to identify environmental conditions. Moreover, the model is capable of representing a description of the sensor, the types of values that it produces and their measurement units. For example, a temperature sensor may return integer values denoting Celsius degrees. Some example statements of the sensor model that define a part of the designed hierarchy of sensor types are the following:

```
sensorType(X) :-
environmentalConditionSensor(X).
environmentalConditionSensor(X) :-
temperatureSensor(X).
temperatureSensor(temperature_sensor).
```

4.2.3. Node Profile

It defines concepts and relationships that refer to the basic features of an IPAC node. It is the core of the metadata models used in the platform. Specifically, in order to express such knowledge, this profile takes advantage of the vocabulary offered by the sensor model. Some metadata belonging to the node profile are the communication interfaces provided by the node, its name, the available storage space and the supported user interfaces.

Furthermore, it provides information about the sensors that are attached to the node. An example of a node profile follows:

```
node(node03).
supportsUI(node03,visual).
hasSensor(node03,sensor05).
hasSensor(node03,sensor06).
hasCommInterface(node03,ieee_802_11_int).
type(sensor05, temperature_sensor).
type(sensor06, smoke_detector).
```

The aforementioned statements describe a node (with ID *node03*) with a temperature sensor (i.e., *sensor05*) and a smoke detector (i.e., *sensor06*) attached. The node also supports visual interfaces and the IEEE 802.11 communication interface, as well.

4.3. Reconfiguration Policies

The IPAC Knowledge Plane also involves some predefined reconfiguration policies that trigger updates in its settings in order to achieve optimal operation of the node. The reconfiguration policies are rules defining changes that could be performed so that the optimal operation of the node is guaranteed. In general, these hard-coded policies aim to prevent the occurrence of unacceptable situations that could deteriorate system performance. Moreover, they may raise events in case the system status is error-prone. Similarly to the application events, the node policies are also represented declaratively, since they constitute part of the overall knowledge base. Two sample reconfiguration policies are:

```
policy(hasCommInterface,X,wisemac_int):-
numberOfNeighbors(X,N), N=0, node(X),
hasCommInterface(X,ieee_802_11_int).

policyPA(turnInterfaceOn,wisemac_int):-
numberOfNeighbors(X,N), N=0, node(X),
hasCommInterface(X,ieee_802_11_int).

policy(hasCommInterface,X,ieee_802_11_int):-
numberOfNeighbors(X,N), N=0, node(X),
hasCommInterface(X,wisemac_int).

policyPA(turnInterfaceOn,ieee_802_11_int):-
numberOfNeighbors(X,N), N=0, node(X),
hasCommInterface(X,wisemac_int).
```

A natural language description for the above set of rules is: "Change the current communication interface in case no neighbours have been detected". Hence, the node should switch from its current communication interface to another (e.g., from IEEE 802.11 to WiseMac). The *policy* predicate defines the new facts that should be asserted and the *policyPA* predicate defines the middleware method call that applies this reconfiguration (in this case the turnInterfaceOn() method of the SRCC Proxy service). In that case, the node profile is updated with the information that no other node has been detected in the vicinity by the node. Since the policies are checked periodically, the next time that such a check will take place, the Reconfiguration service will call the appropriate middleware services (e.g., the SRCC Proxy to modify the communication interface, the User Interaction Service to change the user interface) in order to make the required modification and will also update the node profile.

4.4. Reconfiguration Workflow

The overall approach that has been adopted for the design of IPAC middleware is that applications access directly the middleware services through well defined interfaces (i.e., method calls), but all methods that are responsible for altering the operation/configuration of the services are called through the Reconfiguration Service. The rationale behind this decision is that service and device settings are a shared resource and so it should be managed by a central entity. Moreover, global cross-layer knowledge may be necessary for some reconfigurations. We should not expect applications to maintain such knowledge.

The typical reconfiguration workflow is as follows: An application sends a request for reconfiguration (it may affect a middleware service or the node/device settings). 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 for such requests. Moreover, the applications should not make any assumptions on them. 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 service is invoked. The latter service decides if it is consistent with the current system status to perform the change requested. If not, and,

a) it is a soft request with timeValidity set, then it just reschedules the request and checks it after a preset period of time.

b) it is a hard request, it sends a response "reconfiguration not possible" to the Response Queue.

If the change is possible, a message with the request parameters is sent to the Dispatcher Queue. Then the Reconfiguration Service consumes this message and performs the change requested with a synchronous method call. The result is inserted to the Response Queue (either "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.

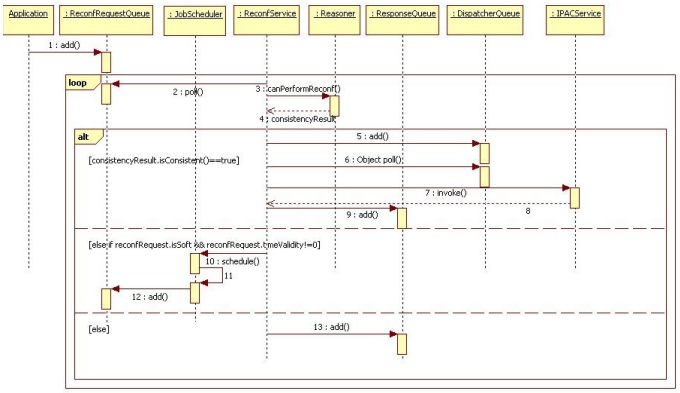

Figure 3. UML Sequence Diagram of Reconfiguration Service

Note that there could also be other services listening to the Response Queue. For example, if we want to provide self-healing functionality, we could observe which services return "Failure" and how often and use this information for recovering from this situation. All this interaction is depicted in the UML Sequence Diagram of Figure 3.

4.5. Reconfiguration Scenario

This use-case scenario describes the main tasks that take place within an IPAC node during the reconfiguration phase. The scenario assumes that application A (e.g., with id *app02*) is stored to an IPAC node (e.g., with id *node03*) in order to be deployed while other applications are already running inside the node and A requires audio interface to run properly (e.g., the statement *requiresUI(app02,audio)* has been added to its profile). The node supports both visual and audio user interfaces. 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).
```

It is also assumed that the following event is defined in the profile of the application A:

```
event(fire_alarm):-smoke_sensor_1>=0,
temp_sensor_1>=20.
```

This rule denotes that an event named "fire_alarm" is triggered when there some substance of smoke in the air and the temperature exceeds the limit of 20 degrees. Additionally, the application A requires a minimum communication range of 10 meters and an audio user interface in order to run (denoted in its profile).

First, the application manager checks whether the requirements of the new application are consistent with the requirements imposed by the applications that already run in the IPAC node. This is performed through the consistency checking mechanism provided by the Reasoner. Assuming that none of the already running applications have imposed any requirements regarding the user interface of the node, it can be switched to audio without any possible conflicts. Hence, a reconfiguration request is scheduled in order to perform the desired reconfiguration action.

Regarding the policy checking mechanism, we assume that a policy denotes that in case there is no other node in the neighbourhood, the communication interface should change (see example in Section 4.3). In that case, the node profile is updated that no other node has been detected close to it through the SRCC. Hence, a policy violation is detected the first time that the checkPolicy() method is called. This method checks if any node reconfiguration can be performed based on the defined policies and returns the respective *ReconfRequest* objects.

The checkPolicy() method invokes the Reasoner service in order to trigger the aforementioned rules (if all of their conditions are satisfied). If triggered, the method parses the asserted *policy* predicates and returns them to the method caller. For example, if the first policy rule is triggered, the following fact is returned (*node03* is the ID of the node):

```
hasCommInterface(node03, wisemac_int).
```

If this function returns some facts, then these are used to create a new *ReconfRequest* object that is subsequently passed as an argument to the checkReconfRequest() method. This method checks if the request is in conflict with the current node configuration. If not, the facts of the ReconfRequest are asserted to the knowledge base and the respective middleware service methods are invoked.

5. IMPLEMENTATION DETAILS

The IPAC Knowledge Plane adopts a Prolog-based scheme in order to support the desired self-adaptive behaviour of the nodes. Hence, all the models, the policies and the application profiles are expressed as Prolog predicates. The implementation of this functionality takes advantage of Java Internet Prolog (JIProlog) [1] engine that constitutes a compact and computationally efficient Prolog interpreter facilitating the integration of Java and Prolog. JIProlog provides a Java API for creating and handling Prolog files. This API supports various management capabilities over knowledge bases that are expressed in Prolog terms (e.g., loading of multiple files, query answering). It is also a cross platform framework that enables the development of lightweight reasoning services on devices with restricted resources. Specifically, the J2ME (MIDP 2.0, CLDC 1.0) version of JIProlog was adopted for the IPAC Reasoner service. This solution performs better than reasoning over other knowledge representation methodologies, such as lightweight ontologies, since no efficient reasoning modules are available for these formalisms in resource constrained devices [6].

6. EXPERIMENTAL EVALUATION

In this section, we discuss the performance of the reconfiguration tasks in typical IPAC nodes. Specifically, certain simulated scenarios were performed in order to quantify the loading times of the application profiles, the service response time and the memory consumption incurred during reconfiguration tasks. All measurements were performed in ASUS Eee PC 900 netbooks [7] with an Intel (R) Celeron (R) processor running at 900MHz and 1GB of main memory. These nodes are able to use both IEEE 802.11 and Wisemac communication interfaces and were equipped with a number of sensor devices to capture context information.

The first measurement concerns the time needed to load the application profiles. While the rest of the models and profiles used (e.g., sensor model, node profile, etc.) can be loaded just once during the initialization of the IPAC platform, the application profiles have to be loaded and retracted at runtime whenever an application starts or stops operating in the node. The experiments demonstrate a linear dependency between the loading times and the number of application profiles. Specifically, the required time is proportional to the number of profiles inserted into the main memory. For example, it is worth mentioning that in case of two profiles, the total loading time is about 15ms. Even when 20 applications were sequentially loaded, the total loading time does not exceed the 125ms which is an acceptable delay in real time applications.

When a new application profile is loaded, the JIProlog engine inserts all the relevant rules and the facts to the main memory. As

a result, the query answering process of the knowledge base is performed very efficiently. For example, even the evaluation of a query containing predicates defined through multiple Prolog rules with several conjunctive terms in their body requires less than 16ms. The tests performed also indicate that such performance persists even when multiple application profiles are loaded to the main memory. Hence, real-time checks for complex queries (e.g., compatibility of a newly deployed application) do not negatively impact the overall response time of the system.

Another issue examined was the sensitivity of the system according to context changes. Specifically, 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). Figure 4 presents the times between a context change that took place and the corresponding node reconfiguration. These times indicate 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 made, (b) the policy execution, (c) the check of the applicability of the requested reconfiguration, and, (d) the execution of the reconfiguration. As described earlier, the execution of policies is performed very efficiently (in the main memory) and the reconfiguration is achieved through simple method calls. Since the policy execution is performed periodically (i.e., not asynchronously) the required time is mostly dependent on the policy checking period.

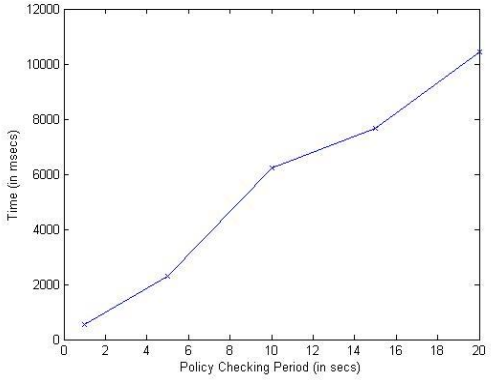

Figure 4. Reconfiguration time in ASUS Eee PC 900 netbook

Figure 4 presents the mean times needed to perform the node adaptation across several policy checking periods. One can observe that these times are approximately the half of the policy checking period used. For example, using a policy checking period of 1 sec, the mean reconfiguration time is 546ms, while setting the period to 20 sec the mean reconfiguration time is 10425 ms. This is anticipated since context changes occur at random. Hence, taking into account that the process of checking the policies does not require a significant amount of time, the policy checking period can be set to a small value (e.g., 1 sec) in order to increase the system throughput.

Finally, we measured the memory consumption of the reconfiguration process depending on the number of the application profiles loaded in main memory (Figure 5). The mean size of each profile was about 4 KBs. When a single application is running in the node the total memory required was 20648 KBs (including all the required services, i.e., the Reasoner, the Scheduler and the Reconfiguration services) while in the case of 20 application profiles the memory needed was measured 21214 KBs. Hence, the mean memory size required for each profile was about 25 KBs. This is caused by the internal data structures and mechanisms used by the JIProlog engine when a new profile is added in order to answer possible queries more efficiently.

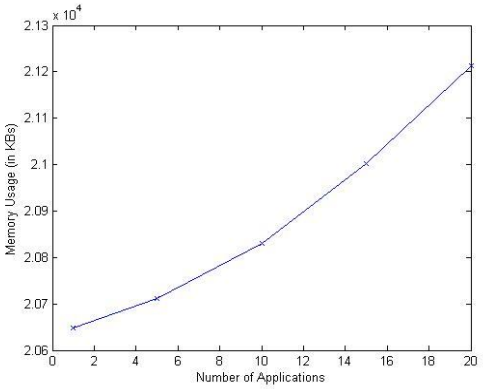

Figure 5. Memory consumption in ASUS Eee PC 900 netbook

7. RELATED WORK

There are several attempts in the areas of autonomic computing and reconfigurable systems aiming to provide adaptive functionality and services. MASS [20] is an ontology-based middleware, aiming to enhance the design, development and provisioning of context-aware applications. It exploits ontologies to express semantic information in mobile devices with limited capabilities, which allows for automated reasoning and adaptation according to user profile and device capabilities. A similar approach is discussed in the Sense Project [21] which focuses on the adaptive behavior of the network processes and management. Furthermore, EMMA Project [22] emphasizes the seamless collaboration of wireless sensing elements in order to achieve intelligent behavior of services. However, none of the aforementioned platforms provides cross-layer reconfiguration of both applications and node settings.

In the area of mobile service intelligence and reasoning, a framework capable of supporting ontology processing is proposed in [4]. The system takes advantage of RDF/OWL ontologies to represent contextual information stemming from sensors and provide reasoning and query answering services over the acquired data. Nevertheless, the system does not investigate the concept of re-configurability since it focuses on the knowledge management processes.

A service discovery framework for mobile music selection is presented in [3]. The system exploits ontological models to capture user preferences and provide personalized discovery of audio content in mobile devices. However, the proposed framework focuses on the concepts of ontology-based relaxation of preferences and semantic matchmaking and does not investigate the re-configurability of the supported applications and devices.

A policy-based information model for mobile ad hoc networks is presented in [17]. Specifically, the proposed solution allows for dynamic reconfiguration of the parameters that affect the routing protocol of the network. Also, a policy-based approach for reconfiguration in autonomous networks is presented in [15]. Such approaches lack extensibility since they are mainly based on hard-coded policies and they focus on specific reconfiguration actions.

8. CONCLUSION AND FUTURE WORK

In this paper we presented a working implementation of a cross-layer reconfiguration framework capable of supporting context-aware applications in autonomic mobile nodes. The framework exploits knowledge management technologies and techniques in order to provide a flexible and extensible solution to node reconfiguration. All node configuration data are expressed through widely-established formalisms, such as Prolog rules and facts. The implemented architecture can support a broad range of reconfiguration scenarios. The functionality and performance of the system was evaluated through deployment on real nodes.

Surely, several aspects of the systems can be improved and are part of our future plans on this area. We intend to use more modern knowledge representation languages for the implementation of the knowledge plane. Specifically, the use of lightweight, but adequately expressive, ontologies will be further investigated. The main problem with using ontologies is the lack of efficient reasoning engines in resource-restricted devices. Since the existing research prototypes [13] have not been tested in real world deployments, we plan to proceed with an assessment of such solutions in the near future.

Another aspect that could be optimized involves the policy checking mechanism of the system. As described, the current IPAC approach checks for possible policy violations periodically. The selection of the appropriate value for this parameter could be optimized by taking into consideration a number of other issues like the history information of the node and the frequency of the policy violations. Event-based mechanisms to detect the context changes that affect the execution of policies should be also explored.

9. 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. The material presented in this article is joint work of the members of the IPAC Consortium (http://ipac.di.uoa.gr).

10. REFERENCES

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

[2] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. 1987. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing* (Vancouver, British Columbia, Canada, August 10 - 12, 1987). F. B. Schneider, Ed. PODC '87. ACM, New York, NY, 1-12. DOI= http://doi.acm.org/10.1145/41840.41841

[3] Noppens, O., Luther, M., Liebig, T. Wagner, M. Paolucci. M. 2006. Ontology supported Preference Handling for Mobile Music Selection , In *Proceedings of the Multidisciplinary Workshop on Advances in Preference Handling*, Riva del Garda, Italy

[4] Gu T., Kwok Z., Koh KK. and Pung HK. 2007. A Mobile Framework Supporting Ontology Processing and Reasoning. In *Proceedings of the 2nd Workshop on Requirements and Solutions for Pervasive Software Infrastructure (RSPS)*, Austria

[5] Tsetsos, V. and Hadjiefthymiades, S. 2009. An innovative architecture for context foraging. In *Proceedings of the Eighth ACM international Workshop on Data Engineering For Wireless and Mobile Access* (Providence, Rhode Island, June 29 - 29, 2009). MobiDE '09. ACM, New York, NY, 41-48. DOI= http://doi.acm.org/10.1145/1594139.1594152

[6] Papataxiarhis, V., Tsetsos, V., Karali, I., Stamatopoulos, P., and Hadjiefthymiades, S. 2009. Developing rule-based applications for the Web: Methodologies and Tools, *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches* (Eds. Adrian Giurca, Dragan Gasevic and Kuldar Taveter), Information Science Reference

[7] ASUSTeK Computer Inc. Eee PC 900 Series, http://www.asus.com/product.aspx?P_ID=PGEqzGdfmtS5rjPk

[8] Sekkas, O., Piguet, D. Anagnostopoulos, C. Kotsakos, D. Alyfantis, G. Kassapoglou-Faist C. and Hadjiethymiades, S. 2009. Probabilistic Information Dissemination for MANETs: the IPAC Approach, in *Proceedings of 20th Tyrrhenian International Workshop on Digital Communications*, Pula, Sardinia, Italy, 375-385

[9] OSGi Alliance. 2004. *About the OSGi service platform. Technical Whitepaper.* Available at http://www.osgi.org.

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

[11] Panayiotou, C., Fytros, E., Tsetsos, V., Samaras, G., Hadjiefthymiades, S., Piquet, D. 2009. Integrated Platform for Autonomic Computing, In *Proceedings of IEEE SECON*, Rome, Italy

[12] El-Hoiydi, A. and Decotignie. J.-D. 2004. Wisemac: An Ultra Low Power MAC Protocol for the Downlink of Infrastructure Wireless Sensor Networks. In *Proceedings of 9th IEEE International Symposium on Computers and Communications*, Alexandria, Egypt.

[13] KRHyper Homepage, http://www.uni-koblenz.de/~wernhard/krhyper/

[14] Parashar M., and Hariri, S. 2006. Autonomic Computing: Concepts, Infrastructure and Applications. CRC Press

[15] Chen, J., Zhao, Z., Qu, D. and P. Zhang. 2008. A policy-based approach for reconfiguration management and enforcement in autonomic communication systems. *Wireless Personal Communications Magazine*, IEEE, 45(2), 145–161

[16] Kim Y. and Kim E., 2006. Autonomic Service Reconfiguration in a Ubiquitous Computing Environment, *Lecture Notes in Computer Science, Parallel and Distributed Processing and Applications*, vol. 4330, 584-593.

[17] DeSiqueira, M., Figueiredo, F., Rocha, F., Martins, J. DeCastro, M. 2005. Policy-Based Dynamic Reconfiguration of Mobile Ad Hoc Networks, *Lecture Notes in Computer Science, Networking - ICN 2005*, vol. 3421, 116-124

[18] Nomikos, V., Kolomvatsos, K., Hadjiefthymiades, S. and Papadopoulos, V. 2009. An Application Creation Environmentfor Autonomic Computing, *2nd Student Workshop on Wireless Sensor Networks*, Athens, Greece.

[19] Mahmoud Q. H., 2007. *Cognitive Networks: Towards Self-Aware Networks*, Wiley

[20] Corradi, A., Montanari, R. and Toninelli. 2007. An Adaptive Semantic Middleware for Mobile Environments, *Journal of Networks*, Academy Publisher, 2, 1

[21] SENSE – Smart Embedded Network of Sensing Entities, www.sense-ist.org

[22] EMMA - Embedded Middleware in Mobility Applications, www.emmaproject.eu

[23] Clark, D. D., Partridge, C., Ramming, J. C., and Wroclawski, J. T. 2003. A knowledge plane for the internet. 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, 3-10. DOI= http://doi.acm.org/10.1145/863955.863957