Resource Management in a Pervasive Computing Environment

Promise A. Nlerum and Edward E. Ogheneovo

1Department of Computer Science and Informatics, Federal University Otuoke, Bayelsa State, Nigeria.
2Department of Computer Science, University of Port Harcourt, Port Harcourt, Rivers State, Nigeria.

Authors’ contributions

This work was carried out in collaboration between the authors. Author PAN designed the pervasive architecture with emphasis on the public-subscribe and resource classifier components. Author EEO performed the ambience calculus analysis and programming abstractions. Both authors read and approved the final manuscript.

Article Information

DOI: 10.9734/AJRCOS/2020/v5i130126
(1) Prof. M. A. Jayaram, Director, Department of Master of Computer Applications, Siddaganga Institute of Technology, Tumakuru, India.
(2) Yulia, Petra Christian University, Indonesia.
(2) Anthony Spiteri Staines, University of Malta, Malta.
(3) Boyan Karapenev, Technical University of Gabrovo, Bulgaria.
Complete Peer review History: http://www.sdarticle4.com/review-history/54523

ABSTRACT

A pervasive computing system provides for the interaction of people, devices and applications in a seamless and transparent manner in a pervasive computing environment. There is a great need for the management of basic resources in this environment. In this paper, we present a resource management architecture that provides for a seamless interaction among pervasive elements using ambient calculus and a publish/subscribe mechanism. Ambient calculus has been used to explore resource interaction and participation in a pervasive computing environment. A resource classifier component is introduced in the architecture that performs resource binding to specific applications. Results show that Ambient calculus offers a convenient and flexible representation of resource availability, usage transparency and management. By incorporating publish subscribe and resource classifier components into the ambient model, our system has shown a high degree of scalability, flexibility, and fault tolerance.

*Corresponding author: E-mail: nlerumpa@fuotuoke.edu.ng;
**Keywords:** Pervasive computing; ambient calculus; usage transparency; fault tolerance; publish/subscribe.

1. **INTRODUCTION**

According to Dan O'Keeffe [1], a pervasive computing environment is a physical space coordinated by a proactive software infrastructure, that enhances the capabilities of applications to configure and adapt their software and physical dimensions, automatically and transparently with respect to the user. Physical environment means the physical dimension of the system. This dimension describes all the physical entities belonging to the environment during the entire lifecycle of the system. As will be discussed in the sections that follow, users are considered to be one of the interacting resources belonging to the environment.

Pervasive computing environments allow different devices and applications to interact, while competing with available computational resources. This poses a problem of how resources are to be managed (in terms of availability to applications and devices) in an environment such as this. Resource hungry applications often times find it difficult to access some resource, even when they are available. The result is that the aim of pervasiveness is lost if the situation persists. There have been significant research works on the efficient management of resources in a pervasive computing environment, however, these works still lack in efficiency and scalability to manage the high traffic and in some cases dearth of resources in the pervasive environment. We have looked at the various strategies employed by different researchers and have designed an architectural framework for resource management and utilization in a pervasive computing environment. We believe the proposed system, which incorporates a publish-subscribe mechanism into the architecture will relieve the infrastructure of a central resource management and hence ensure flexibility, efficiency and scalability.

Software infrastructure provides the interconnection framework for the elements of the environment and a coordination model to allow these elements to cooperate on a common goal. An application represents the functional aspect of the system. An application is structured in two parts: Interaction and application logic. Interaction defines the interchange between the elements of the environment and the system. The application logic handles the notion of services and their coordination [2]. Unlike traditional desktop applications with a graphical user interface, pervasive computing forces us to take a rather holistic view of a system. In traditional GUIs, the interactive system is the desktop computer and a fixed set of input/output devices. The emphasis is on combining software components to provide services to the user. Pervasive computing on the contrary refers not only to software services but also to devices and how to describe their interactions.

According to [3], mobility is an important aspect of pervasive computing; pervasive elements are allowed to move to and fro the pervasive computing environment. The concept of pervasive computing began, not as an exercise in using the Internet, but as a study of how people work, how they use tools and where the future of computing might lie. The objective of pervasive computing is to move computers away from the central focus of the user's attention into the invisible world, where they are used subconsciously, to enhance existing tools or communications, and to make the user free from time and space constraints. So, the goal of pervasive computing is the production of devices that are so commonplace and natural to use that they become almost invisible. The need for an efficient resource management mechanism cannot be overemphasized in a pervasive computing environment. Services are embedded in resources hence the need for efficient tracking and management of resources. The resources/devices being pervasive means there will be hundreds of small computers in an office or home, each doing its own specialized task [4].

2. **RELATED WORK**

Al-Muhtadi et al. [5] embedded services into resources with the aim of developing a distributed middleware infrastructure that coordinates software entities and heterogeneous networked devices contained in a physical space. The physical space is termed as an active space in which users interact with several resources and services at the same time. Gaia exports services to query access, and uses existing resources and provides a framework to develop user centric mobile applications.
Jay Ligatti et al. [6] used a domain-specific language, which they called LoPSiL to create a resource management model that detects devices and resources. They claim that LoPSiL is the first expressive (Turing-complete) policy-specification language that targets location-based policies. LoPSiL’s novelty is that it provides abstractions for conveniently accessing and manipulating location information in policies, which is a valuable tool for resource management and allocation. Due to the popularity of Java (particularly Java ME) as an application programming language for mobile devices, LoPSiL was implemented in the context of Java.

Want and Garlan [7] in their Aura pervasive computing project, proposed a programming model for task-based computing. They modelled resources as task based elements. In this model, resources are viewed as compositions of services. Both tasks and services have explicit representations. Services are described by virtual service types, which define functional, state and configuration interfaces and dependencies upon other services. Virtual service types can be related through inheritance, and can also be composed to form new virtual services. Tasks are top level compositions of services that are specified as flows that decompose tasks into steps of subtasks or primitives (actions carried out by services).

Preuveneers and Berbers [8] proposed a self-adapting middleware to optimize resource usage in a component-based application. The main difference with our work is that they do not consider heterogeneous platforms and their middleware runs inside a Java Virtual Machine, we also do not modify or build an application depending on a specific programming model. Instead, we have presented resources to the existing applications in a transparent manner.

Whitaker et al. [9] in their Denali project, virtualizes each hardware resource as a component to share with each guest operating system. In our proposal we look for pervasive resources and apply similar concepts of virtualization using ambience, so the system and the applications use virtual resources without noticing if they are remote or local.

Jakob [10] introduced the concepts of Resource Monitor and the Context Actuator. A monitor is a client specially designed for monitoring resources and acquiring context information in the environment by cooperating with some kind of sensor equipment, and associate it properly with an entity. A context actuator is a client designed to work together with one or more actuators to affect or ‘change’ the context. The Java Context Aware Framework (JCAF) framework can handle the acquisition and transformation of context information in two modes. In the asynchronous mode resource monitors constantly deliver context information to one or more context services, which then can notify listeners or be queried.

In [11], it was showed how context templates could be used to create context patterns that constrain contexts and define resource information relevant within a particular application. The context template defines context structures with valid attributes and values. The validation mechanism of the Context Middleware ensures that a given context instance always is valid toward the context template for a domain.

Maffioletti et al. [12] identified the high degree of dynamism and heterogeneity of the resources involved in a pervasive computing environment, which makes service adaptation and interoperability a difficult task. They presented a system called UBIDEV; a service framework that faces the heterogeneity problem by hiding at the application level the dynamism of the underlying environment. UBIDEV architecture focuses on the description and the management of services and resources. They also described how this approach decreases the complexity of the design and development of service-oriented applications.

3. FORMAL MODEL OF PERSUASIVE COMPUTING ENVIRONMENT

Pervasive computing aims to support and enhance human activity through the use of a spectrum of computation and communication resources. The support of different kinds of activities is hence an important property of pervasive computing environments. These activities can range from meetings, classes or seminars in university or office environments to patient guidance and monitoring in hospital or home-care environments to vehicle tracking and route finding in roads, etc. [13]. Since the support of activities is an integral part of pervasive computing environments, we make it a central piece of our model as well.
One of the key questions that we will attempt to answer with our model is given a description of an activity, A, and the description of a pervasive computing environment, E, which contains a set of resources, R and which is in a certain context, can the environment E support the activity A? In addition, does the environment satisfy certain important properties during the process of performing the activity? Fig. 1 shows a block diagram that describes a sub-activity model of a pervasive computing environment. We have also included a resource classifier component in the architecture. The Resource Classifier answers most of the above posed questions.

The Resource Classifier ensures that each activity acquires the required resource(s) for its operations. Context state is being achieved using the classifier module. Its main task is to adapt an application to a context.

According to [14] the specification and verification of pervasive computing environments involves the following steps:

1. Description of the pervasive computing environment and the resources it contains using ambient calculus.
2. Description of the kinds of operations that the environment can perform in terms of ambient calculus actions.
3. Description of the activities that the environment should support.
4. Definition of any properties that the environment should satisfy.
5. Verification of whether various configurations of the environment do indeed support the activities, while satisfying the desired properties.

According to [15], any pervasive computing environment has a certain set of resources at a given point of time. Each resource has some properties and functionality and may be associated with a context. In order to perform a task, the environment needs one or more suitable resources. In order to use a resource for performing a task, the resource must be capable of performing the task and it must be in an appropriate context.

The Resource Classifier ensures that applications are not starved of resources. We have modelled the resource classifier as composing of the following components as shown in Fig. 2.

The resource classifier uses two adaptation mechanisms to classify context for an application; the Dynamic Adaptation and the Decision Adapter. The application uses rule-based reasoning for the Decision Adapter and the publish/subscribe mechanism for the Dynamic adaptation of resources to applications. We have simulated the classifier using the above context pattern.

![Fig. 1. Formal model of a pervasive computing environment](image-url)
4. RESOURCE MANAGEMENT

Resources include all elements of the environment that are manipulated by the application [16]. Resources come from multiple, heterogeneous devices, such as Laptops, PDAs, Sensors, Phones, network devices etc. Heterogeneity in pervasive computing systems is not meant to disappear in the future, but instead will increase as the range of computing devices increase. Devices in a pervasive computing environment will include sensors and actuators that mediate between physical and virtual environments; embedded devices in objects such as watches and shoes; home and office appliances such as videos, toasters and telephones; mobile devices, such as handheld organizers and notebooks; and traditional desktop machines. Heterogeneous devices will be required to interact seamlessly, despite wide differences in hardware and software capabilities. This will require an infrastructure that maintains knowledge of device characteristics and manages the integration of devices into a coherent system that enables arbitrary device interactions (for example, between a mobile phone and a desktop workstation).

If we consider devices used by the user to interact with the system, they can range from standard ones such as laptops, PDAs, and phones, to emerging ones such as those embedded in clothing and eyeglasses. The variety of available devices has several implications. One is the kind of input-output devices: Textual and graphic input-output will not be the only forms of human-machine interaction. Audio, visual, and other sensory modes of communication will be prevalent. Another implication is the requirement that the environment must be prepared to adapt to the device currently used by the user. For example, if the user is requesting information and is currently driving, the retrieved data should be relayed to him with an audio message through the car radio [17].

Fig. 3 shows a typical resource manager architecture, adopted from our work on Architecture for pervasive computing systems.

Component Handler (CH): This module acts as a Component Loader and is responsible for component launch (configuration and registration), maintenance of component runtime information and maintenance of component life cycle. When the Loader launches a component instance, it adds the information of the component instance to the runtime information table. At initialization, all components are recognized and available to the client.

Messaging (inter/intra): The messaging module provides services for components to send and receive messages over arbitrary message delivery mechanisms such as the informative Bus or sockets. The messaging module is built on a modular, plug-in architecture for the flexibility and extendibility of the messaging systems.

Resource Discovery (RD): The discovering of new components in the environment is the main task in this module. RD also is built on a modular, plug-in architecture for the flexibility and extendibility of the discovery system.

Resources Monitoring (RM): This module keeps track of the current devices’ status, informing to the Component Handler when a new device appears or disappears.

Publish/Subscribe (PS): Publishes the available components in the system or it subscribes to required components in nodes. Only subscribed components can participate in the computational processes. This way, the resource manager is relieved of unnecessary devices in the pervasive environment, a way of improving device and resource interaction.
5. AMBIENCE CALCULUS FOR PERVERSIVE COMPUTING SYSTEMS

Ambient calculus was proposed by [18], to describe the movement of processes and devices. An ambient is a bounded place where computation happens. A key property of ambients is the existence of a boundary around an ambient. The boundary determines what is inside and what is outside an ambient. Ambient boundaries define the scope of a computation and therefore establish a container, which may be easily identified. Examples of ambients are: A web page (bounded by a file), a virtual address space (bounded by an addressing range), a Unix file system (bounded within a physical volume), a single data object (bounded by self) and a laptop (bounded by its case and data ports).

Ambients can be nested within other ambients to form a tree-based hierarchy of containers within other containers, [19]. An example of this is a Java applet executing within a web page, which runs within a browser, which runs on an operating system, which runs on a laptop. Each ambient has a name, and an ambient can be moved as a whole, into and out of other ambients, by performing in and out operations. Ambient calculus allows describing complex phenomena in terms of creation and destruction of nested ambients, and movement of processes into and out of these ambients.

Anand and Roy [20] proposed a formal model for describing pervasive computing environments based on ambient calculus and the associated ambient logic. The model provides a flexible platform to state and verify several properties of these environments such as “anywhere anyhow services”, “mobility of devices and applications” and “context-aware adaptation”. The model also allows the description of the resources present in an environment, the operations that can be performed in the environment, and how users can use the resources in the environment to perform different kinds of activities.
The following subsection provides greater detail about the modelling afforded by ambient calculus.

5.1 Mobility Primitives of Ambient Calculus

In [21] an important requirement for pervasive computing systems is the ability to adapt themselves at runtime to handle such things as user mobility, resource variability, changing user needs, and system faults. A process is an instance of a program under execution. Runtime is captured as and ambient and all processes therein.

Table 1 lists the mobility primitives defined by Ambient Calculus. In ambient calculus, all entities are described as processes. An important kind of process is an ambient. n[P] represents an ambient, where n is the name of the ambient, and P is the process running inside the ambient. An ambient can be thought of as a bounded region (physical or virtual) that can contain processes.

The process 0 is a process that does nothing. Parallel execution is denoted by a binary operator “|” that is commutative and associative. !P denotes the unbounded replication of the process P. That is, !P can produce as many parallel replicas of P as needed, and is equivalent to P | !P.

Ambients can be nested.

For example, the expression n[P1 | ... | Pj | m1[...] | ... | mk[...]] describes an ambient with the name, n, that contains j processes with the names P1, ..., Pj and k ambients with the names m1 ... mk. Some processes can execute an action that changes the state of the world around them. This behavior of processes is specified using capabilities.

Table 1. Key mobility primitives of ambient calculus

| P ::= | process |
| n[P ] | ambient |
| 0 | inactive process |
| P |Q | composition of processes |
| !P | replication of a process |
| M,P | capability action |
| (x).P | input action |
| (M ) | asynchronous output action |
| (ν n)P | restriction, creation of name n |
| M ::= | capability |
| in M | can enter M |
| out M | can exit M |
| open M | can open M |
| x | variable |
| n | name |
| M,M' | composition of capabilities |

The process M,P executes an action described by the capability M, and then continues as the process P. There are three kinds of capabilities: one for entering an ambient, one for exiting an ambient and one for opening up an ambient. Fig. 4 shows these three capabilities.
The process in m.P instructs the ambient surrounding in m.P to enter a sibling ambient named m. If no sibling m can be found, the operation blocks until a time when such a sibling exists. If more than one m sibling exists, any one of them can be chosen. The reduction rule, which specifies the change in state of the world is: n[in m.P | Q] | m[R] → m[n[P | Q] | R]

The action out m.P, instructs the ambient surrounding out m.P to exit its sibling ambient named m. If the parent is not named m, the operation blocks until a time when such a parent exists. The reduction rule is:

m[n[out m.P | Q] | R] → [P | Q] | m[R]

The action open m.P provides a way of dissolving the boundary of an ambient named m located at the same level as open. If no ambient m can be found, the operation blocks until a time when such an ambient exists.

The rule is:

open m.P | m[Q] → P | Q

An output action, releases a capability (possibly a name), of the form, <M>, into the local ether of the surrounding ambient. An input action of the form, (x).P, captures a capability, x from the local ether and binds it to a variable within a scope. An input action corresponding to one process and an output from a different process can be composed as below:

(x).P | <M> → P {x ← M}

The restriction operator (ν n)P creates a new (unique) name n within a scope P. The new name can be used to name ambients and to operate on ambients by name.

### 5.2 Ambient Logic

According to [22] Ambient Logic is a kind of modal logic that allows talking about properties that hold at certain locations. Logical formulae in ambient logic include those defined for propositional logic (like truth, negation and disjunction), spatial operators that are based on ambients (such as composition and location), temporal operators (such as eventually and always), and quantifications (universal and existential). Various model-checkers have been proposed for ambient logic, that allow automatic verification of properties described in ambient logic.

Table 2 lists the primitive formulas that can be expressed using Ambient Logic and what is required for a process to satisfy a certain formula. The satisfaction relation P |= A means that the process P satisfies the closed formula A (i.e. A contains no free variables).

The satisfaction relation is defined inductively in Table 2, where Π is the sort of processes, Φ is the sort of formulas, Ν is the sort of variables, and Λ is the sort of names.

In the temporal modality, the relation P → P^t indicates that the ambient P can be reduced to P^t using exactly one reduction rule in [23]. Then, P →* P^t is the reflexive and transitive closure of the

<table>
<thead>
<tr>
<th>Primitive Formulas</th>
<th>Reduction Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀p ∈ Π</td>
<td>p</td>
</tr>
<tr>
<td>∀p ∈ Π [a ∈ φ]</td>
<td>p = ¬a</td>
</tr>
<tr>
<td>∀p ∈ Π [a, b ∈ φ]</td>
<td>p = a ∨ b</td>
</tr>
<tr>
<td>∀p ∈ Π</td>
<td>P = 0</td>
</tr>
<tr>
<td>∀p ∈ Π [a, b ∈ φ]</td>
<td>P = P^t</td>
</tr>
<tr>
<td>∀p ∈ Π [a, b ∈ φ]</td>
<td>P = a ∧ P^t</td>
</tr>
<tr>
<td>∀p ∈ Π [a, b ∈ φ]</td>
<td>P = ν[n]</td>
</tr>
<tr>
<td>∀p ∈ Π [a ∈ φ]</td>
<td>P = a@n</td>
</tr>
<tr>
<td>∀p ∈ Π [a, b ∈ φ]</td>
<td>P = a@P</td>
</tr>
<tr>
<td>∀p ∈ Π [a, b ∈ φ]</td>
<td>P = ν[φ]</td>
</tr>
<tr>
<td>∀p ∈ Π [x ∈ N, a, b ∈ φ]</td>
<td>P = νx</td>
</tr>
</tbody>
</table>

Table 2. Satisfaction in ambient logic
reduction relation. Similarly, in the spatial modality, the relation \( P \downarrow P \) indicates that the ambient \( P \) contains \( P \) within exactly one level of nesting. Then, \( P \downarrow\ast P \) is the reflexive and transitive closure of the previous relation, indicating that \( P \) contains \( P \) at some nesting level [24].

5.3 Ambient Programming for Resource Management

In this section, we demonstrate an experiment that shows how AmbientChat reconciles the expressive power of a high-level language with the ability to cope with the difficulties engendered by the hardware phenomena described in previous sections. The experiment involves the implementation of what we call ambient resource monitoring (ARM), which can be thought of as a kind of network reference which is stable with respect to resource discovery. The Publish/Subscribe module provides for the relief on total resource management of the pervasive infrastructure.

The following is a sub algorithm and program implementation of our architecture.

5.3.1 Algorithm

```
ALGORITHM: PUBLISH_SUBSCRIBE RESOURCE
// Identify resource in pervasive environment
Publish_table := null_empty
// Store resource using hashtable
Resource := Hash_table
// Tag an ID to resource
ResourceID := ID_resource
// Subscription by applications and devices: Applications loop
Search_Hash_table := get_resource // searches resource hash_table
Resource := Resource_ID // Assign ID to resource
// Resource functions are also emulated to provide for resource hungry applications
Request_clone := clone_lookup
If found Resource.clone else discard request.clone
// Attach clone request to specific applications
Join.observe := resource_application
// delete after completion
Request.delete := resource.Application
// Iterate as pervasive ambient is active
Iterate loop
```

5.3.2 Code extract

```
Code extract
publish_resource_msg :: actor({
resource : void; identity : void;
cloning.new(id) :: {
resource := Hashtable.new();
identity := id;
subscribe.add(id)
};
address_source(resource_Id) :: {
resource.put(resource_Id,AmbientRef<-new(resource_Id)));
sendMessageTo(resource_id,text) :: {
resource.get(resource_id)<-receive(identity,text)
};
receive(from, text) :: {
display(from," : ",text)
};
AmbientRef :: actor({ ref : void;
cloning.new(tag) :: { required.add(tag);
in.addObserver(this.onReceive);
joined.addObserver(this.onJoin);
disjoined.addObserver(this.onDisjoint)
});
```
6. CONCLUSION

We have defined the ambient-calculus paradigm and an ambient programming abstraction as a set of programming language characteristics that directly deals with the hardware phenomena encountered when developing applications for resource monitoring and management. Our architecture, which combines an ambient model with a publish/subscribe mechanism tracks relevant changes in the pervasive environment. The implementation of this architecture shows that our architecture is efficient and result oriented. The resource classifier component, embedded in our architecture ensures proper resource assignment to applications. The system sieves the unnecessary from the whole, improving scalability and performance. We have evaluated our approach by means of the implementation of an example scenario in Java. In this implementation the programmer clearly needs to spend most of his time managing the connectivity with little emphasis on the resource management. The Publish/Subscribe module and the resource classifier components provide for robustness of our architecture, as they monitor the ambient environment carefully, ensuring that resources are assigned the appropriate application and devices.

COMPETING INTERESTS

Authors have declared that no competing interests exist.

REFERENCES


10. Jakob EB. The Java Context Awareness Framework (JCAF) – a service


