

Provably Correct Pervasive Computing Environments

Anand Ranganathan
IBM T.J. Watson Research Center
Hawthorne, NY 10532
arangana@us.ibm.com

Roy H. Campbell
University of Illinois at Urbana-Champaign
Urbana, IL 61801
rhc@uiuc.edu

Abstract

The field of pervasive computing has seen a lot of exciting innovations in the past few years. However, there are currently no mechanisms for describing the properties and capabilities of pervasive computing environments in a formal manner. This makes it difficult to prove the correctness of a pervasive computing environment, i.e. to verify that the environment satisfies certain desired properties. In this paper, we propose a formal model for describing pervasive computing environments based on ambient calculus and the associated ambient logic. The model allows us 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 allows us to describe 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. As a case study, we shall describe some of the resources and operations supported by the Gaia middleware using this model, and verify an example property of a pervasive computing environment supported by Gaia.

1 Introduction

Pervasive computing aims to provide computing and communication services all the time, everywhere, transparently and invisibly to the user, using devices embedded in the surrounding physical environment. In the past few years, a number of pervasive computing systems and environments have been developed and tested. These systems use a variety of novel ap-

proaches to provide useful functionality to end-users. However, it is often difficult to describe the properties and features of these systems in a precise, formal manner. One reason for this is that rapid evolution of these systems has, so far, not been supported by formal methods that allow the description and verification of the properties and features of these environments. This makes it difficult to prove, in a formal manner, that a certain pervasive computing environment does satisfy certain properties, i.e. it operates “correctly” according to certain specifications.

The problems of precisely specifying the properties of pervasive computing environments and verifying them become especially important as pervasive computing moves into critical domains such as healthcare, vehicle control and traffic management. In these domains, it is essential to ensure that the design of the environment is correct, that it will behave as per expectations, and that it always satisfies important properties like availability, reliability, adaptability and security. As the size of the environments grow, with increasing numbers of entities (devices, services and users), it becomes impossible to verify their properties manually. In such cases, automated means of verification is necessary to check all possible interactions between the different entities and ensure that they satisfy the properties of interest.

In order to prove the correctness of pervasive computing environments, we take recourse to techniques employed in formal verification. In the formal verification literature, different models have been proposed for specifying the behavior and properties of software and systems. Examples of objects used to model systems include finite state machines, labeled transition systems, Petri nets, timed automata, pi calculus, pro-

cess algebra, etc. Using these models, the properties of systems can be verified using model checking techniques. The properties are often specified using variants of temporal logics.

In the case of pervasive computing environments, a key aspect to model is location. Many desired properties of these environments deal with mobility of different entities and location-sensitive behavior. One model that fits the bill in this regard is *ambient calculus*. Ambient calculus has been recently proposed as a theoretical framework for distributed and mobile objects/agents [5]. An ambient is a bounded place where computation happens, and hence has a strong notion of location (either physical or virtual location). Ambient Calculus provides some primitive operations for describing the movement, replication, creation and dissolution of ambients. It is a very expressive calculus and has been shown to be Turing complete [5].

Associated with Ambient Calculus is Ambient Logic [4]. Ambient Logic allows stating logical formulas using both temporal and spatial operators. Many pervasive computing properties like “anywhere services” and “mobility of devices and applications” are based on a concept of location. Hence, Ambient Logic is suitable for describing these properties and for verifying if a certain pervasive computing environment does satisfy these properties.

The key contributions of this paper are:

- A method for modeling various kinds of resources, such as devices, services and users, in pervasive computing environments using ambient calculus.
- A method for specifying operations that can be performed and events that can occur in a pervasive computing environment using ambient calculus.
- A method for describing and verifying the properties of pervasive computing environments using ambient logic.

As a case study, we describe various resources and operations supported by Gaia [8] using this model. Gaia is a middleware for enabling Active Spaces, which are pervasive computing environments, where physical spaces have been enhanced with a large number of digital devices such as sensors, computers and

actuators. We have deployed Active Spaces in a number of rooms in the Computer Science building at the University of Illinois, Urbana-Champaign. Our experiences in designing and maintaining these Spaces over a period of several years have convinced us of the need for formal methods for purposes of modeling and verification. Our experiences have also helped us to identify several useful properties of these Spaces, which we shall describe in this paper. As part of the case study, we shall describe the model of a sample Active Space, as well as an example desired property of this Active Space. Finally, we shall provide a proof procedure based on ambient logic for verifying this desired property in the sample Active Space.

2 Background

2.1 Ambient Calculus

Ambient calculus was proposed by Luca Cardelli and Andrew Gordon 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. 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. The following subsection provides greater detail about the modeling afforded by ambient calculus.

Table 1. Ambient Calculus: Mobility Primitives

$P ::=$	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
$\langle M \rangle$	asynchronous output action
$(\nu n)P$	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

2.2 Mobility Primitives of Ambient Calculus

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 $n[P_1 | \dots | P_j | m_1[\dots] | \dots | m_k[\dots]]$ describes an ambient with the name, n , that contains j processes with the names P_1, \dots, P_j and k ambients with the names m_1, \dots, m_k .

Some processes can execute an action that changes the state of the world around them. This behavior of processes is specified using *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 ambi-

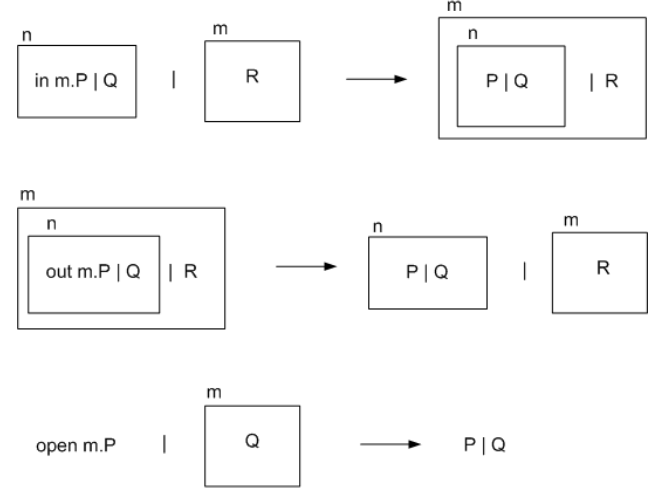


Figure 1. Ambient Calculus: Primitive capabilities

ent, one for exiting an ambient and one for opening up an ambient. Fig 1 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:

$$n[in\ m.P | Q] | m[R] \rightarrow m[n[P | Q] | R]$$

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:

$$m[n[out\ m.P | Q] | R] \rightarrow n[P | Q] | m[R]$$

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:

$$open\ m.P | m[Q] \rightarrow P | Q$$

An output action, releases a capability (possibly a name), of the form, $\langle M \rangle$, 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:

$$(x).P | \langle M \rangle \rightarrow P \{x \leftarrow M\}$$

The restriction operator $(\nu n)P$ 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.

2.3 Ambient Logic

Ambient Logic is a kind of modal logic that allows talking about properties that hold at certain locations. Logical formulas in ambient logic include those defined for propositional logic (like true, 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 $P \models A$ 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, N is the sort of variables, and Λ is the sort of names.

In the temporal modality, the relation $P \rightarrow P'$ indicates that the ambient P can be reduced to P' using exactly one reduction rule. Then, $P \rightarrow^* P'$ is the reflexive and transitive closure of the 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, $P \downarrow^* P'$ is the reflexive and transitive closure of the previous relation, indicating that P contains P' at some nesting level.

3 Formal Model of Pervasive Computing Environments

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.

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 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?

The specification and verification of pervasive computing environments using our model 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.

In our model, we further break down an activity into a set of tasks. For example, the activity of a meeting in a smart conference room may involve a number of tasks like displaying a presentation on one or more devices, allowing one or more users in the room to control the presentation, adjusting room settings (like the light and sound), etc. For the environment to support such an activity, it must provide mechanisms by which these constituent tasks may be performed.

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.

Table 2. Satisfaction in Ambient Logic

$\forall P \in \Pi$	$P \models T$	
$\forall P \in \Pi, a \in \Phi$	$P \models \neg a$	$\cong \neg P \models a$
$\forall P \in \Pi, a, b \in \Phi$	$P \models a \vee b$	$\cong P \models a \vee P \models b$
$\forall P \in \Pi$	$P \models 0$	$\cong P \equiv 0$
$\forall P \in \Pi, a, b \in \Phi$	$P \models a b$	$\cong \exists P', P'' \in \Pi, P \equiv P' P'' \wedge P' \models a \wedge P'' \models b$
$\forall P \in \Pi, a, b \in \Phi$	$P \models a \triangleright b$	$\cong \forall P' \in \Pi, P' \models a \Rightarrow P P' \models b$
$\forall P \in \Pi, n \in \Lambda, a, b \in \Phi$	$P \models n[a]$	$\cong \exists P' \in \Pi, P \equiv n[P'] \wedge P' \models a$
$\forall P \in \Pi, a \in \Phi$	$P \models a@n$	$\cong n[P] \models a$
$\forall P \in \Pi, a, b \in \Phi$	$P \models \diamond a$	$\cong \exists P' \in \Pi, P \rightarrow^* P' \wedge P' \models a$
$\forall P \in \Pi, a, b \in \Phi$	$P \models \nabla a$	$\cong \exists P' \in \Pi, P \downarrow^* P' \wedge P' \models a$
$\forall P \in \Pi, x \in N, a, b \in \Phi$	$P \models \forall x.a$	$\cong \forall m \in \Lambda, P \models a\{x \leftarrow m\}$

For example, in order to display a presentation, the space needs devices with displays along with applications that can display slides on these devices. 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.

In order to ground the subsequent discussion, we describe the model of a specific kind of pervasive computing environment, viz. an Active Space, which is defined as a physical space augmented with rich computation and communication devices. In the following sections, we describe the model of Active Spaces [8] and the resources within it, a model of activities, and a formal definition of what it means for an Active Space to support an activity.

3.1 Model of Active Space

An Active Space is modeled as an ambient in the Ambient Calculus. In addition, the various resources contained within an Active Space as well as the context of the space are also modeled as ambients.

3.1.1 Model of Resources

Resources in an Active Space include software components, devices, users, physical objects and other Active Spaces. Software Components include different kinds of services and applications that may be run in an Active Space. They may communicate using different mechanisms like CORBA, Web Services, Jini, etc. Examples of services are discovery services, services that provide the current context, authentication services,

etc. Examples of applications are music-playing applications, slide-presentation applications, etc. Devices are hardware entities in an Active Space. They may be infrastructure devices present in the environment (like sensors, wall displays and desktops), or user's personal devices (like cellphones, laptops, PDAs and wearable devices). Users are also modeled as being resources in an Active Space, since the performance of an activity may be tied to different users in the Active Space performing different kinds of tasks. Physical Objects include chairs, tables, and other objects in an Active Space. Finally, Active Spaces may contain other Active Spaces. For example, the Siebel building for Computer Science in our university has an Active space containing Laboratories that are also Active spaces.

Since we model all resources as ambients, we can now express spatial relationships between different resources in a tree-like manner. For example, we can now say that a device contains one or more software components; a user contains one or more devices; a space contains devices, software components, physical objects, users, etc. As an example, a certain device *dev* may contain two files, *file1* and *file2*, and four software components. These software components include two applications *app1*, and *app2*, and two services, *serv1* and *serv2*. In this case, this device ambient, *dev* may be represented as:

$$dev[file1 | file2 | app1[P] | app2[Q] | serv1[R] | serv2[S] | Z]$$

In the above description, *P* represents the process running inside the ambient *app1*. *P* captures the contents (or the current state and context) of the application *app1*. Similarly, *Q*, *R* and *S* capture the contents of the ambients *app2*, *serv1* and *serv2* respectively. The

process Z captures all the other contents of the device dev . Z itself may be the composition of other processes, some of which may be ambients. For instance, Z may include other files and software components.

We model context using a process in the ambient calculus. This allows us to associate a context with a certain ambient. For example, a user may have many personal devices and may be associated with a certain context, $cxt1$.

$user1[dev1[P1] | dev2[P2] | cxt1]$

Devices $dev1$ and $dev2$ may further be modeled as before, i.e. they may contain various files, software components, etc. $P1$ and $P2$ may be the composition of different processes representing files, software components, etc. The context $cxt1$ may be a composition of different kinds of contexts. For example, it may include a location of the user, his availability for instant messaging, and his position:

$cxt1 \cong Room3201 | available | sitting$

An Active Space may contain many devices and users and may be associated with a certain context, as below (and in Fig 2):

$as1[dev3[file3 | app3[R3] | P3] | dev4[P4] | user5[dev5[P5] | cxt5 | Q5] | user6[Q6] | cxt2 | Z]$

The structure of an Active Space gives a lot of information about how different devices, users and other ambients relate to one another. For example, devices that are directly inside the Active Space ambient are public devices, while devices that are inside user ambients are personal devices that can only be used by the user. Similarly, software components that are located within a user ambient belong to the user, and can only be used by the user, unless the user gives explicit permission to someone else to use it. The Active Space can also contain physical objects like lights, which can have their own state (e.g. off or on). Note that it is not necessary to model every single file and component and object. We only need to model those that are relevant to the aim of creating the model (e.g. for proving some property).

Apart from the ambients described above, an Active Space can contain other ambients to represent the logical structure of devices, software components and users. For example, if a group of users are working together in a collaborative manner using a subset of devices and software components, this set of ambients (the users, software components and the devices) can

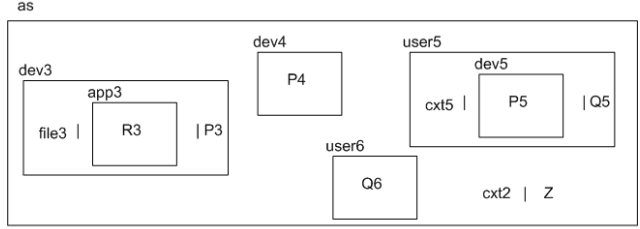


Figure 2. An example Active Space ambient

be represented as being inside an ambient.

Different processes in an Active Space have different capabilities. For example, a device may be capable of hosting some kinds of components, but not capable of hosting others. A certain PDA (called $pda1$), for instance, may be able to run a component to control the slides of a presentation called $slideControlComponent$, but may not be able to run a computationally intensive faceRecognition component called $faceRecognizer$. Such kinds of properties are expressed using ambient logic:

$pda1 \models canHost(slideControlComponent)$

$pda1 \not\models canHost(faceRecognizer)$

The above assertions describe the ambient $pda1$ as satisfying or not satisfying certain predicates, viz. the $canHost$ predicates. Similar assertions can be made about other ambients. For example, we can describe the component $slideControlComponent$ as allowing the performance of a slide control task:

$slideControlComponent \models canPerform(slideControl)$

3.2 Operations in an Active Space

All operations in an Active Space can be modeled as ambients. The different operations involve different resources in the environment, events sent in the environment and changes in context. An example operation is starting a software component in a device. The operation is invoked by composing the operation ambient with the resource ambient. This composition typically results in a reduction process where certain processes move across ambient boundaries. We describe a few such operations in this subsection. As an example, we go into depth of how the first operation (instantiating a component on a device) can be modeled using the primitive Ambient Calculus operators (in, out and open). The other operations can be modeled similarly.

1. The action (or operation) of a component (P) being instantiated on a device (dev) can be represented as:

start in dev.P

There are many ways of describing the *start* operation using the primitive ambient calculus operators. One possible way is to create a temporary ambient, k , that represents a message which is guided into the device, dev . This message ambient also contains the component P inside a special ambient (*deviceEntry*). This special ambient is recognized by the device, and acts as a key that is required to start the component in the device.

$start\ in\ dev.P \cong (\nu k)k[in\ dev.\ deviceEntry[out\ k.\ open\ k.\ P]]$

A device can give explicit permission for a component to be instantiated in it by having a process like *!open deviceEntry* within it. This process allows the device to extract the component from within the special ambient, *deviceEntry*. When the above action ambient is invoked by composing it with a suitable device, we get the following reduction:

$start\ in\ dev.P \mid dev[Q \mid !open\ deviceEntry]$
 $\rightarrow (\nu k)k[in\ dev.\ deviceEntry[out\ k.\ open\ k.\ P]] \mid dev[Q \mid !open\ deviceEntry]$
 $\rightarrow (\nu k)dev[Q \mid !open\ deviceEntry \mid k[deviceEntry[out\ k.\ open\ k.\ P]]]$
 $\rightarrow (\nu k)dev[Q \mid !open\ deviceEntry \mid open\ deviceEntry \mid deviceEntry[open\ k.\ P] \mid k[]]$
 $\rightarrow dev[Q \mid !open\ deviceEntry \mid P]$

A component is said to be running if and only if it is within a device ambient (since all components have to run on a device, and cannot run in a vacuum). Hence, at the end of the above reduction, the component P is running on the device dev . The above description of the action has the feature that it requires the device to give permission for component instantiation; hence it provides one layer of security.

2. The action of stopping a component (P) running on a device (dev) can be represented as :

stop in dev.P

3. The action of a device entering an Active Space (e.g. by either powering it up or installing it within the space) can be represented as:

enterDev in as.dev

4. The action of a device leaving an Active Space (e.g. by either switching it off or after a failure) can be represented as :

exitDev out as.dev

5. The action of a user entering an Active Space can be represented as an objective movement of the user into the Active Space. After this action, the user becomes a part of the space:

enterUser in as.user

6. The action of a user leaving an Active Space can be represented as an objective movement of the user out of the Active Space:

exitUser out as.user

7. The action of a user in an Active Space making a device available can be represented as an objective movement of the device out of the user ambient and into the surrounding Active Space ambient. After this action, the device becomes part of the Active Space ambient; hence, it becomes a public device and other users and services can make use of this device.

moveDev out user.dev

8. Events are represented using asynchronous output actions (such as $\langle E \rangle$). An event can then be consumed by another process. Each event has a certain scope, which is the ambient it is in. Only other processes in that ambient can consume the event. The action of sending an event to any ambient (a device, a user, an Active Space or a software component) can be represented as

send in a. $\langle E \rangle$

9. Change in context of any ambient can be represented as a composition of two ambients, one that removes the original context ($cxt1$) from the ambient and one that adds the new context ($cxt2$) into the ambient. The operation may be represented as:

changeContext in a.[remove[cxt1] | add[cxt2]]

3.3 General Design Principles behind the Model

Some of the key design ideas of our model for Active Spaces are :

- All entities in the space are represented by a tree-based topology of boundaries.
- Mobility of any kind of object (device, user, software component, etc.) is represented as crossing of boundaries.
- Security is represented as the ability or inability to cross boundaries. Special processes are used to represent keys that can be used to open ambients to allow processes to cross boundaries.

- Interaction between processes is scoped by the ambients they belong to. For example, if all personal devices are modeled as being contained with the ambients of users, then a device within one user’s ambient cannot directly interact with a device within another user’s ambient. All interactions have to go through the ambients of the two users.

3.4 Model of Activity

As described earlier, an activity consists of a set of tasks. To capture this relationship between an activity and the tasks within it, we model an activity as an ambient, and the set of tasks as a composition of processes. For example, a meeting activity may involve the tasks of displaying a presentation, controlling the presentation and having the lights dimmed. In this case, the activity is represented as:

$$mtg[displayPresentation \mid controlPresentation \mid dimLights]$$

We now describe how we model tasks. In general, in an Active Space, a task is performed by running certain kinds of software components on certain kinds of devices. Hence, a task is often parameterized by a device and a component. For example, the *displayPresentation* task may be performed by running a component that can display slides on a suitable device. Using the high-level operators described earlier, this task may be modeled as

$$displayPresentation \cong (c).(d).start \text{ in } d.c \text{ such that } \\ c \models isComponent \wedge canPerform(slideDisplay) \text{ and } d \models isDevice \wedge canHost(c)$$

The above assertion states that the *displayPresentation* process takes in two inputs, c and d , and then starts the component, c , in the device d , where c satisfies the predicate $canPerform(slideDisplay)$, i.e. c allows the task of displaying slides, and d can host the component c .

There may be many possible choices of devices and components for accomplishing the *displayPresentation* task. The choice of the device may also be influenced by the context of the devices and the users. We can add new predicates to the *displayPresentation* ambient definition to take the context into account.

We introduce a new high-level operation for performing an activity in an Active Space: *perform in as.activity*. This operation results in

the contents of the *activity* ambient moving inside the Active Space ambient, *as*, if the Active Space allows the entry. The reduction rule is:

$$perform \text{ in } as.activity[P \mid as[Q \mid !open \text{ entryIntoAS}]] \rightarrow^* as[P \mid Q \mid !open \text{ entryIntoAS}]$$

Our next task is to verify that the activity can indeed be performed in the Active Space. For this purpose, we write a formula in ambient logic that describes properties that must be satisfied by the Active Space to allow the performance of the activity. For example, if the Active Space is to allow the performance of the meeting, there must be some component within the Active Space that controls the slides, some component that displays the slides and all the lights in the space must be dimmed.

$$performMeeting = \nabla canPerform(slideControl) \wedge \nabla canPerform(slideDisplay) \wedge (\forall d(as[d] \wedge isLight(d)) \Rightarrow dim(d))$$

Hence, we need to verify if composing the *perform* activity operator with the Active Space leads to the *performMeeting* predicate becoming true eventually:

$$perform \text{ in } as.activity \mid as \models \diamond performMeeting$$

The above formula will hold true if the Active Space has appropriate devices and components that will allow the different tasks in the activity to be performed.

3.5 Properties of Active Spaces

One use of the formal model is that we can specify various properties that Active Spaces must satisfy. In this section, we give examples of some useful properties of Active Spaces and show how they can be expressed.

1. Anywhere, Anyhow activities. This is the property that a certain activity can be performed in all spaces that belong to a given set of Active Spaces. If Σ is a set of Active Spaces, an activity is anywhere and anyhow, if

$$\forall as \in \Sigma \ perform \text{ in } as.activity \mid as[Q] \models \diamond performActivity$$

where *performActivity* is a formula in ambient logic that captures the ability of the activity to be performed.

2. Device Mobility. This is the ability for users to bring in devices into the environments and for those devices to be used in performing certain kinds of tasks. This property may be modeled as the movement of a *user* having a device *dev1* into an Active Space,

as , eventually resulting in the Active Space having an event with the name of the device, $devI$. This event can then be consumed by a task that is parameterized by the device.

$$\begin{aligned} & \text{enterUser in } as.\text{user}[devI \mid P] \mid as[Q] \models \\ & \diamond as[\nabla \text{event}(devI)] \\ & \text{where } \langle devI \rangle \models \text{event}(devI) \end{aligned}$$

3. Context-Sensitivity or responding to changes in context. This is the ability for an Active Space to reconfigure itself after a change in context so that certain properties are still maintained. For example, a useful property of an Active Space is that the device used for performing a certain task is one that is closest to the user. Then, we can verify that if the context changes (e.g. the user moves), the Active Space can reconfigure itself so that the device chosen is still the one that is closest to the user.

In a general case, let us say that a property p needs to be preserved by an Active Space. If $as[cxt1 \mid Q] \models p$, then we need to verify that

$$\begin{aligned} & as[cxt1 \mid Q] \mid \text{changeContext in } a.[\text{remove}[cxt1] \mid \text{add}[cxt2]] \\ & \models \diamond p \end{aligned}$$

4 Case Study based on the Gaia middleware for Active Spaces

One of the properties of Gaia is that it allows a user to bring in his personal devices (like PDAs and cell-phones) into an Active Space and use it for performing various kinds of tasks (like viewing slides or controlling music playing applications). In this section, we shall describe some of the related operations of Gaia and show how this property can be verified.

When a *user* enters an Active Space, as , he first has to log into the space using one of various mechanisms [1]. If the login is successful, any device he is carrying (such as a *pda*) sends a heartbeat to the Gaia Discovery Service, ds , giving it's location as an ambient ($\text{in } user.\langle pda \rangle$). The user entry operation is, thus, modeled as:

$$\begin{aligned} & \text{enterUser in } as.\text{user}[pda \mid P] \\ & \cong (\nu k)k[\text{in } as.\text{loginAS}[\text{out } k.\text{open } k.\text{user}[pda \mid P] \mid \\ & \text{in } ds.\langle \text{in } user.\langle pda \rangle \rangle]] \end{aligned}$$

It can be shown that

$$\begin{aligned} & \text{enterUser in } as.\text{user}[pda \mid P] \mid as[Q \mid \text{!open loginAS} \mid ds] \rightarrow^* \\ & as[Q \mid \text{!open loginAS} \mid ds \mid \text{user}[pda \mid P] \mid \text{in } ds.\langle \text{in } user.\langle pda \rangle \rangle] \end{aligned} \quad (1)$$

The Gaia Discovery Service, which is located within the as ambient is modeled as a process that takes in a heartbeat and sends out an advertisement to the appropriate ambient where the discovered component can be used.

$$\text{Gaia Discovery Service} = ds[!(c).\text{out } ds.c]$$

Using this definition of the Discovery Service, (1) reduces to

$$as[Q \mid \text{!open loginAS} \mid ds \mid \text{user}[pda \mid P \mid \langle pda \rangle]] \quad (2)$$

Also, (2) $\models \diamond as[\nabla \text{event}(pda)]$ which is the condition for device mobility as described in Section 3.5. There is now an event within the user ambient advertizing the *pda*. Hence, any task that is specifically meant for this user can bind to this device. Suppose the task, *showSlide* is to use a specific component like a Slide-Viewer on any suitable device that the user is carrying. $\text{showSlide} \cong mv \text{ in } user1.((d).\text{start in } d.\text{SlideViewer})$ such that

$$d \models \text{isDevice}(d) \wedge \text{canHost}(\text{SlideViewer})$$

Here, mv is an operation that instructs one ambient to move into another. It can be seen that $\text{perform in } as.\text{activity}[\text{showSlide}] \mid as[R \mid \text{user}[pda \mid P \mid \langle pda \rangle]] \rightarrow^* as[R \mid \text{user}[pda[\text{SlideViewer}] \mid P]]$

That is, the user's PDA now has the *SlideViewer* application instantiated on it. Hence, the user can perform a task using the device.

5 Related Work

The main modeling work, so far, in pervasive computing has centered around modeling context, e.g. using ontologies. However, there has not been much work in describing models of the entities and the behavior of pervasive computing environments as a whole. Birkedal et al [2] have done some initial work in using bigraphical reactive systems to model certain aspects of pervasive computing. Weis et al [9] suggest a programming paradigm for pervasive applications based on ambient calculus. These efforts, however, have not yet tackled the problem of describing and verifying properties of the environments.

We see the ontology-based models for describing context as complementary to our ambient calculus model. Ontology-based models help in defining the vocabulary of use, as well as for inferring higher-level contexts, while ambient calculus helps in defining the

behavior and processes within the environment. The two may be combined in interesting ways to describe both the vocabulary and the behavior of the environment, although this is beyond the scope of this paper.

6 Conclusion and Discussion

In this paper, we have described a basic framework for modeling pervasive computing environments and the activities that can be performed in them using Ambient Calculus. In addition, we have shown how various properties of the environments can be stated and verified using Ambient Logic. The model is highly extensible to cover a wide range of possible behaviors and properties of Active Spaces.

There are a number of ways in which the formal model can be used. It can be used as an early design tool of new pervasive computing environments. It allows describing an environment formally and verifying properties about the environment even before the environment is prototyped or deployed. It can also be used as a technique for verifying properties about existing environments and comparing the properties of different environments. Using this technique, one can distinguish environments based on the resources, the operations that can be performed and the activities that can be supported. Finally, it can be used as a guide for generating test cases for monitoring different entities and verifying that they do indeed satisfy the properties during runtime.

One of the limitations of formal verification arises from the gap between the specification and the actual implementation. A formal specification describes the intended behavior of the environment. It would require significant work to verify the correctness of a particular environment and its middleware (such as Gaia), during runtime, in terms of this model. Nevertheless, formal verification still has its place in checking systems, because if the specification of the system is found to be incorrect, then the actual implementation is almost certain to behave incorrectly. Hence, formal verification plays an important role in guarding against critical design errors.

In this paper, we have verified properties by direct proofs. More complex spaces, with many resources, and many concurrent activities will require the use of a model checker. A model checker is an algorithm that

determines the truth of the assertion $P \models a$, given process P and formula a as input. Many model checkers for ambient calculus have been proposed ([3],[7],[6]) for different fragments of ambient logic. Verification of properties may require the use of additional reasoning depending on how context and other features of Active Spaces are modeled. For example, depending on the model of context used (like ontology-based, fact-based and spatial), we may need to invoke different kinds of reasoning on these models (rule-based, DL-based, relational algebra, etc.) to verify some of the properties.

References

- [1] J. Al-Muhtadi, A. Ranganathan, R. H. Campbell, and D. Mickunas. Cerberus: A context-aware security scheme for smart spaces. In *IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, Dallas-Fort Worth, March 2003.
- [2] L. Birkeedal, M. Bundgaard, T. C. Damgaard, S. Debois, E. Elsborg, A. J. Glenstrup, T. Hildebrandt, R. Milner, and H. Niss. Bigraphical programming languages for pervasive computing. In *International Workshop on Combining Theory and Systems Building in Pervasive Computing, part of PERSASIVE 2006*, 2006.
- [3] L. Cardelli and A. Gordon. Anytime, anywhere modal logics for mobile ambients. In *27th ACM Symposium on Principles of Programming Languages*, 2000.
- [4] L. Cardelli and A. D. Gordon. Ambient logic. *Mathematical Structures in Computer Science*.
- [5] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science, Special Issue on Coordination*, 240(1):177–213, June 2000.
- [6] W. Charatonik and J.-M. Talbot. The decidability of model checking mobile ambients. In *15th Annual Conference of the European Association for Computer Science Logic*, 2001.
- [7] W. Charatonik, S. D. Zilio, A. Gordon, S. Mukhopadhyay, and J.-M. Talbot. The complexity of model checking mobile ambients. In *Foundations of Software Science and Computation Structures (FOSSACS'01)*, 2001.
- [8] M. Roman, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Narhstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing Magazine*, 1:74–83, 2002.
- [9] T. Weis, C. Becker, and A. Brandle. Towards a programming paradigm for pervasive applications based on the ambient calculus. In *International Workshop on Combining Theory and Systems Building in Pervasive Computing, part of PERSASIVE 2006*, 2006.