

# Towards a Programming Paradigm for Pervasive Applications based on the Ambient Calculus

Torben Weis<sup>1</sup>, Christian Becker<sup>1</sup>, and Alexander Brändle<sup>2</sup>

<sup>1</sup> Universität Stuttgart, Univeritätsstrasse 38, Stuttgart  
{weistn, beckercn}@informatik.uni-stuttgart.de

<sup>2</sup> Microsoft Research, Cambridge  
alexbr@microsoft.com

**Abstract.** In this paper we suggest a new programming paradigm for pervasive applications that is based on the ambient calculus. We believe that a paradigm shift is required, because object-oriented design has several shortcomings when applied to pervasive computing. Most notably, a set of objects is usually heavily entangled because objects hold references to many other objects and perform synchronous method invocations on each other. If some objects vanish because their host has weak connectivity or is simply turned off, it is hard to predict the behavior of the system. Furthermore, security issues are hard to deal with in object-oriented modeling. We show that distributed pervasive applications can be modeled using ambients, processes, and asynchronous communication. We discuss the difference to object-oriented modeling and show that ambient-based modeling can greatly improve the design of pervasive applications. Finally, we introduce  $N\#$  - a new programming language for pervasive applications based on the ambient paradigm.

## 1 Introduction

Pervasive systems are very dynamic and involve a large set of devices connected by different networking technologies. Since the involved devices are end-user devices, users tamper with them in unforeseeable ways. Thus, devices are not as reliable as PCs or even servers. Furthermore, the networking techniques used are more error prone. To cope with such an ever changing and unreliable environment, applications must track changes of the environment and adapt accordingly.

Approaches for tracking changes of the environment can be divided in two classes. The first class builds or maintains a model of the real world (i.e. context services) on which applications act. Examples are ontologies [1, 2], location models [3, 4], and taxonomy based conceptualizations [5, 6] that provide means to relate objects in the physical world to each others. This allows to express that two objects are nearby, that a person is a participant in a meeting, etc. The second class of approaches pushes context information and its structure directly into the application logic, i.e. no model of the real world is maintained.

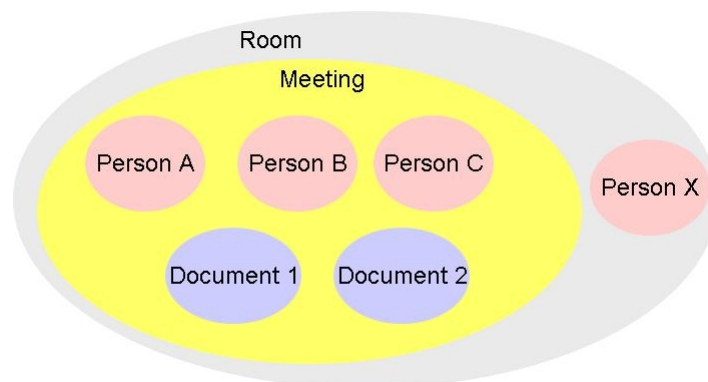
---

\* The presented work has been funded by DFG Excellence Center 627 "Nexus" and Microsoft Research Cambridge

A prominent example is the Context Toolkit [7]. It resembles a collection of components that form a pipeline. This pipeline processes and modifies context information retrieved from sensors. Applications register callbacks to consume context events.

Most architectures for pervasive applications put the burdon of adaptation entirely on the shoulders of the developer, e.g. one.world [8] and GAIA [9]. Others address adaptivity by decoupling application parts via events, e.g. iROS [10]. However, there are no means to automatically adapt the application based on changes of the environment. Furthermore, it is hard to analyze security issues because security boundaries are not made explicit by current mainstream programming paradigms. In this paper we argue that we need a programming paradigm based on ambients. To underpin this claim, the following section highlights the deficiencies of object-oriented programming when applied to pervasive applications. In the third section we argue that the ambient calculus is a useful foundation for a new pervasive computing programming paradigm. Then, we discuss mobile agents and how they can be used to implement the ambient calculus. Finally, we introduce our pervasive computing programming language N#, which is based on the ambient calculus.

## 2 Criticism of Object-Oriented Design

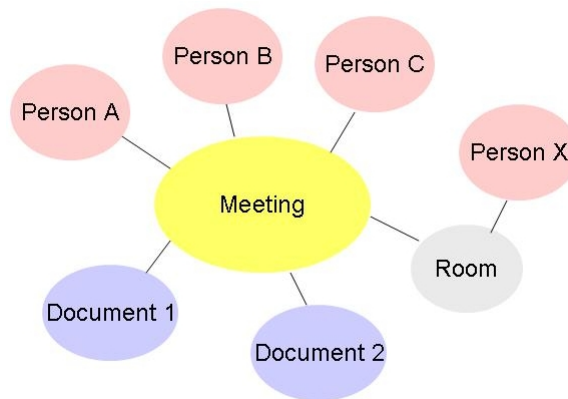


**Fig. 1.** Context-specific relationships

Figure 1 shows a room where a meeting takes place. Persons A, B, C attend the meeting and have access to Documents 1 and 2 that are available in the meeting. Person X, also in the room, does not participate in the meeting and has no access rights to the documents. We see a number of relationships that are represented here. First, a spatial relation. The meeting takes place in a room. Persons are also located in that room. This is expressed by an inclusion, represented by the room ellipse that contains the meeting and the persons also

represented as ellipses. Second, there is a relation indicating that a person attends a meeting. This is also represented as an ellipse representing the meeting that contains the persons. The meeting is embedded in the spatial inclusion hierarchy. The beforementioned access rights directly correspond to the hierarchies. Thus, person X being in the room may adjust the light but may not access the documents of the meeting.

Figure 2 shows the same scenario, but this time it is based on an object-oriented design. Thus, Figure 2 shows a set of objects which constitute a flat graph. The nodes of this graph are objects and object references are represented as edges. The concept of containment is entirely missing. Thus, by looking at the diagram, it is not immediately visible that the documents are not meant to be accessible for "Person X". Of course object-oriented implementations can somehow enforce this policy. This is not the important point. The point to make is that a language used to express context-aware applications should make important information *explicit*.



**Fig. 2.** The example modeled with objects

Furthermore, a programming language should enable the developer to express application logic in a concise way. For example, imagine that "Person A" publishes a new "Document 3" to be available for all others in the meeting. Using an object-oriented approach, the document must be linked with the "Meeting" object and the "Meeting" object must notify all connected "Person" objects about the new document. Doing this reliably and secure in the presence of distribution, network errors, and a snoop "Person X" is already a non-trivial task. Using a language that treats context-specific relationships as first class entities, the developer could simply state something like

```
new document_3( some_data ).out
```

to create the "Document 3" and move it out and into the "Meeting", i.e. the outer ellipsis. It is now the task of the compiler to generate an implementation that manages the document and notifies all meeting members.

This example demonstrates that a clever selection of programming language concepts can drastically simplify the application development. Furthermore, the implementation is more concise, easier to understand and maintain. Currently, it is an open question which concepts are central to context-aware applications. However, the concept of a containment hierarchy based on context-specific relationships seems very promising and it lines up well with the ambient calculus [11, 12] as discussed in the next section.

### 3 Applying the Ambient Calculus

An ambient is a bounded place where computation happens. A very intuitive illustration of the ambient calculus can be found in [12] (see Figure 3). In this illustration, each ambient is depicted as a folder. This folder has a name and processes are executing inside the folder. Special operations such as in, out, and open allow folders to become subfolders, leave a parent folder, or to remove a folder. Figure 3 illustrates the basic operations of the ambient calculus.

This fits nicely with the above example of meetings, rooms, persons, and documents. Every element of the real world can be treated as an ambient. Since computation happens inside an ambient, we can easily express that functionality is bound to places, situations, persons, or physical objects. To represent a meeting and associated functionality, the developer simply creates an ambient and implements the functionality as processes running inside the ambient. Furthermore, the boundaries of an ambient can be treated as security boundaries. To see what is inside an ambient (i.e. to see which documents are available during a meeting) you must first enter the ambient. Thus, we can treat Figure 1 as an ambient calculus in action by calling the ellipses ambients. We believe that the ambient calculus can be used as a theoretical foundation for a context-aware programming language. However, calculi in general are not meant to be used as a programming language. The ambient calculus does not (and does not have to) feature any convenient means for mathematical computation or to control legacy software (PowerPoint, MediaPlayer) or electrical devices (light, roller blinds, air conditioning etc.).

It is the goal of our research to create a convenient language that combines the elegant concepts of the ambient calculus with traditional programming concepts. Ideally, the new language is compatible with existing legacy systems, i.e. it should be possible to use existing software components. The approach is to allow processes inside an ambient to use features of the underlying platform and operating system. Thus, the language strictly controls the borders of the ambient to protect privacy and to avoid hidden dependencies between ambients. In contrast, the language allows the interior of ambients to be implemented using traditional programming concepts in addition to ambient-operations such as in, out, and open.

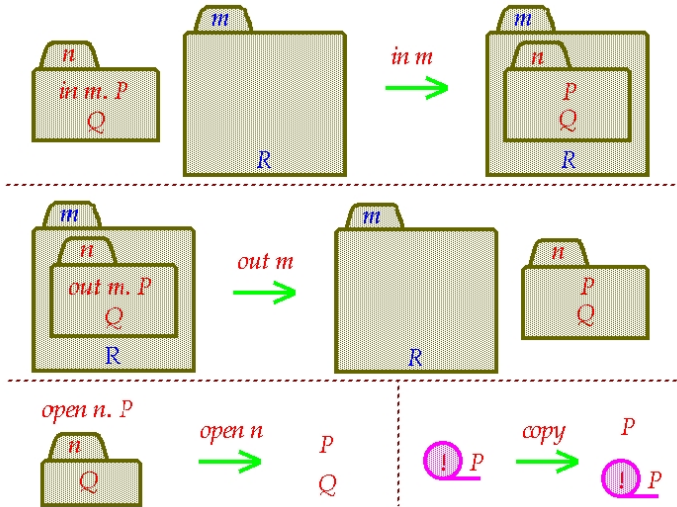


Fig. 3. Ambient calculus primitives (taken from [12])

### 3.1 Mobile Agents

Some languages based on the ambient calculus have been presented [13,14]. They follow the idea of mobile agents, i.e. every ambient is considered to be a mobile agent. Moving an ambient  $n$  inside an ambient  $m$  means that  $n$ 's implementation and state are serialized, sent to  $m$ , deserialized, and resurrected. Unfortunately, this approach has limitations in practice. None of the presented solutions became famous for efficiency in terms of bandwidth, memory, and CPU usage because migrating ambients causes high network load and spends much CPU time on serialization and deserialization. In pervasive applications, many devices are rather small resource restricted devices. There is hardly a realistic chance of deploying a mobile agent system on them. Imagine that every object in the real world (building, room, bottle, book) is represented by a mobile agent that migrates from one machine to the other as the real object moves. This way, taking a shopping basket from the shop to your home would already cause a significant network load due to many mobile agents migrating from the shop computer to the computer built into your fridge. Furthermore, synchronization is a non-trivial endeavour for distributed implementations of the ambient calculus, because the three operations in, out, and open are atomic and involve up to three ambients. These ambients must synchronize. The JoCAML-based [15] mobile agent implementation published in [13] is the first distributed and asynchronous implementation that does not require a global lock. However, there is no forceful reason to implement the ambient calculus on top of mobile agents. We must clearly distinguish between means for thinking about an application and means for executing it. This leads directly to the next section.

### 3.2 N# - A Programming Language based on Ambients

We are currently working on N#, a new programming language for pervasive applications. N# combines the concepts of the ambient calculus with a syntax that closely resembles mainstream languages such as C# or Java. The N# compiler must always have access to a context-server, because the meta-model used by the context-server is integrated with N#'s type system. Thus, if the context server knows about persons participating in meetings, we can create ambients bound to meetings and persons.

The following code snippet creates a "MyMeeting" ambient for every meeting detected by the context-server. Therefore, the ambient is bound to "Meeting", which is a type defined in the context-server meta-model. The ambient features a named port "InstantMessagePort". Everybody inside the meeting can post a message on this port or wait for messages to arrive on this port. Thus, if one person sends a message via this port, all others can receive the message. The important point to make here is that the port is not accessible to people outside the meeting.

Inside the meeting ambient, an ambient for every person participating in the meeting is created. For this to work, the context-server meta-model must know the type "Person" and the "Meeting" type must feature a role named "Participant". Every person ambient features a "ReceiveMessage" process which listens on the meeting's "InstantMessagePort". Furthermore, the "Send Message" process is connected to a graphical user interface. The GUI itself is not implemented in N#. Instead, we use C# to build a DLL implementing the GUI and link it with the N# application.

```
ambient MyMeeting @ Meeting
{
  port InstantMessagePort;

  ambient Participant @ Person
  where role(Participant in super)
  {
    process SendMessage @ msg = GUI.Message
    {
      post(InstantMessagePort, msg) }
    }

    process ReceiveMessage
    {
      select m = !wait(super.InstantMessagePort)
      {
        print("The message is: " + m);
      }
    }
  }
}
```

In the N# implementation you can see immediately from the source code who is entitled to receive instant messages published inside a meeting. This would not be the case for an object-oriented implementation. In N# all communication between ambients and processes is asynchronous. This is a direct result from the ambient calculus. We adopted the concept of named ports from the  $\Pi$ -calculus to simplify the syntax. In the pure ambient calculus you would create a new message ambient inside a participant, move it out in the meeting ambient, copy it there and move the copies inside the person ambients. Obviously, that would be too troublesome for developers. Therefore, we created this shortcut.

Furthermore, N# uses declarative programming in many places. For example, you do not need to write code that queries the context-server for persons entering or leaving the meeting. This is done implicitly by the N# runtime. Thus, if a new person enters the meeting, a "Participant" ambient is created. If the person leaves the meeting, the ambient is opened (i.e. terminated) automatically.

The major difference to mobile agent implementations is that N# ambients do not migrate from one machine to another. To illustrate this, we image a snapshot of ambients depicted as a tree. Edges denote parent-child relationships between ambients and nodes denote ambients. In the mobile agent approaches the nodes can migrate from one computer to another. In N#, the nodes do not move, only edges (i.e. parent-child relationships) are inserted or deleted.

## 4 Conclusions and Outlook

We suggested the ambient calculus as a formal foundation for a new pervasive computing programming paradigm. Based on some examples we could show that object-oriented approaches make important aspects such as security *implicit*, while ambient calculus-based modeling can make these aspects *explicit*. Finally, we introduced our new programming language N#, which is based on the ambient calculus. Ambients are a way for N# developers to structure their application. However, the generated implementation does not move ambients from one computer to the other. Thus, ambients are a pure design-time constructs in N#. In contrast, mobile agent approaches uphold the ambient abstraction even at runtime, which results in higher bandwidth and CPU usage.

N# is still under heavy construction, but a prototype compiler already exists. In the future, we plan to implement a set of pervasive applications in N# (ambient-oriented) and in C# (object-oriented) to compare both approaches. Furthermore, we will adapt our graphical modeling tool VisualRDK [16,17] to N#. This way, even hobby programmers will be able to produce pervasive applications using a convenient graphical programming language based on the ambient calculus.

Finally, we will have to investigate, whether the ambient calculus is too strict for very dynamic pervasive systems. We have to find means of dealing with errors and lost messages. Furthermore, the strict synchronization of the three ambient operations could become a bottleneck in practice.

## References

1. Henricksen, K., Indulska, J.: Modeling context information in pervasive computing systems. In: Proceedings of 1st International Conference on Pervasive Computing (Pervasive 2002). (2002)
2. Chen, H., Finn, T., Joshi, A.: Semantic Web in the Context Broker Architecture. In: Proceedings of International Conference on Pervasive Computing and Communication (PerCom 04). (2004)
3. Becker, C., Duerr, F.: On location models for ubiquitous computing. *Personal and Ubiquitous Computing* **9** (2005) 20–31
4. Judd, G., Steenkiste, P.: Providing Contextual Information to Pervasive Computing Applications. In: Proceedings of International Conference on Pervasive Computing and Communication (PerCom 03). (2003)
5. Nicklas, D., Mitschang, B.: On building location aware applications using an open platform based on the nexus augmented world model. *Software and System Modeling* **3** (2004) 303–313
6. Schmidt, A., Beigl, M., Gellersen, H.W.: There is more to context than location. In: Proceedings of the Intl. Workshop on Interactive Applications of Mobile Computing (IMC98). (1998)
7. Dey, A.K., Abowd, G.D., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction* **16** (2001) 97–166
8. Grimm, R., Davis, J., Lemar, E., MacBeth, A., Swanson, S., Anderson, T., Bershad, B., Borriello, G., Gribble, S., Wetherall, D.: System support for pervasive applications. *ACM Transactions on Computer Systems* **22** (2004) 421–486
9. Roman, M., Campbell, R.: Gaia: Enabling active spaces. In: 9th ACM SIGOPS European Workshop. (2000) 229–234
10. Ponnekanti, S., Johanson, B., Kiciman, E., Fox, A.: Portability, Extensibility and Robustness in iROS. In: Proceedings of International Conference on Pervasive Computing and Communication (PerCom 03). (2003)
11. Cardelli, L., Gordon, A.: Mobile Ambients. In: FoSSaCS 98. Volume 1378 of LNCS. (1998)
12. Cardelli, L.: Mobile Computational Ambients. <http://www.luca.demon.co.uk/Ambit/Ambit.html> (2006)
13. Fournet, C., Levy, J.J., Schmitt, A.: An Asynchronous, Distributed Implementation of Mobile Ambients. In: Proceedings of IFIP TCS 2000. Volume 1872 of LNCS. (2000)
14. Fallah-Seghrouchni, A.E., Suna, A.: CLAIM: A Computational Language for Autonomous, Intelligent and Mobile Agents. In: Proceedings of PROMAS 2003. Volume 3067 of LNCS. (2004)
15. Conchon, S., Fessant, F.L.: JoCAML: Mobile Agents for Objective-CAML. In: First International Symposium on Agent Systems and Applications (ASA99). (1999)
16. Weis, T., Handte, M., Knoll, M., Becker, C.: Customizable pervasive applications. In: Proceedings of the International Conference on Pervasive Computing and Communications (PerCom2006). (2006)
17. Ulbrich, A., Weis, T., Geihs, K.: A Modeling Language for Applications in Pervasive Computing Environments. In: 2nd Workshop on Model-based Methodologies for Pervasive and Embedded Software. (2005)