Good Manners for Pervasive Computing – an Approach based on the Ambient Calculus

Gregor Schiele*, Marcus Handte+, and Christian Becker*

*Universität Mannheim
Schloss, 68131 Mannheim, Germany
{gregor.schiele | christian.becker}@uni-mannheim.de

+Universität Stuttgart
Universitätsstrasse 38, 70569 Stuttgart, Germany
marcus.handte@ipvs.uni-stuttgart.de

Abstract

When people interact, they follow distinct rules that coordinate the order of speech, who opens doors, whom and how to greet, and many things more. Such a social codex depends on the milieu – or ambience – people are acting in. People breaking the codex are either considered badly educated or foreigners to the ambience – sometimes even both. In Pervasive Computing a multitude of applications is expected to populate our environment and to follow objects and users throughout their daily journey. Consequently, we will need a new codex – or manners – for Pervasive Computing applications that controls the interaction between applications. Such a codex will have to incorporate our existing codices as well as technical aspects. In this work in progress paper we present an approach to extend our prior work on Pervasive Computing system support by specifying interdependencies of applications based on the ambient calculus. This allows specifying and technically enforcing "manners" for Pervasive Computing applications.

1. Introduction

Pervasive Computing aims at providing distraction-free support for users in their everyday environment. To do so, Pervasive Computing applications typically are context-aware in the sense that they detect the user's current physical context and adapt their behavior to it. As an example, an application may sense that a user is currently driving in his car and switches from visual to audio output. In another scenario, the application may detect that the user is standing nearby a large display and redirect its output to it. Developing such adaptive applications is a complex and error-prone task. Different approaches to help developers handling this task have been proposed (e.g. [1], [9], [6], [8], [7]). They typically provide a model to structure an application as a number of cooperating entities, e.g. software components, and a runtime environment to execute such applications. However, applications are often viewed in isolation and are expected to coexist peacefully. In reality, a lot of users will execute a multitude of applications concurrently in the same Pervasive Computing environment. These applications will interfere with each other, both in their resource requirements and their user's goals. To handle such interferences, applications need to coordinate themselves and agree on ways to solve the conflicts. One way to do so is to define a set of application spanning rules which define how the system should react to conflicts and how a consistent system state can be achieved. We call this set of rules a codex. Applications following a certain codex are considered well behaved. In the following we give some examples for interferences between applications and discuss simple codices to solve them.

2. Interferences and Codices

As motivation consider a simple "follow-me" application that plays the user's favorite music and utilizes appropriate audio gear in the user's vicinity. While moving from the bedroom to the bath room, Anne listens to her favorite morning show. Meanwhile, Bob is in the living room, chatting with a friend on the phone using a hands-free speaking system. When Anne enters the room, her music application

detects the audio equipment in the environment and redirects the music to it, interfering with Bob's phone call. To detect and handle such interferences the developer of the music application needs to foresee and handle this situation manually. Using our system, Anne could define a codex rule for her music application, stating that the application should tune down the music if it is executed in the same environment as another user's application which uses audio, too.

Other examples for interferences are applications competing for the same resources. If e.g. both Anne and Bob want to listen to music, the system must decide, which radio channel to switch to. To solve this question, a codex rule could define an ordering between users, stating, e.g., that Bob allows Anne's goals to take precedence over his own. Alternatively, the codex could specify that the users themselves must determine a conflict resolution and present the users with a possibility to enter their decision into the system.

In addition, a certain class of privacy issues can be seen as interference between user goals. While one user reads his private email on a large wall-mounted display, another user walks into the room. This violates the first user's goal to read his email privately, i.e. without any other user in line of sight of the used display. The system automatically detects this interference between the users and switches the application's output to another display to solve it.

Lastly, conditions on the physical surroundings can be defined using codex rules. As an example, a logistics application can use such rules to prevent placing reacting chemical components close to each other.

3. System Model

Our system consists of multiple devices that are either located in the environment, e.g. wall-mounted displays or sensor nodes, or carried by users, e.g. personal digital assistants or smart clothing. The devices are highly heterogeneous concerning their resources, e.g. computational power or memory. All devices are networked spontaneously, e.g. using some wireless communication technology, and form a mobile ad hoc network (MANET).

In the system, multiple users access the devices to execute adaptive applications, i.e. applications that adapt themselves to changes in the environment. We assume the presence of suitable system software for adaptive applications, e.g. [1].

4. Requirement Analysis

Our goal is to resolve interferences between applications of different users automatically. Three steps are necessary to do so. The first step is to specify possible interferences. This is necessary not only for detecting them but also for presenting them to users. For the specification, a suitable notation is needed, e.g. one of the notations given in [4].

In the second step, interferences must be detected at runtime. Interferences emerge, because applications influence the physical context of their users in conflicting ways, e.g. because both use an audio output component. Therefore, changes in the physical context must be monitored and checked against the interference specifications created in step one.

The third step in handling interferences is to resolve them. This is done by providing codices which specify valid resolution strategies for different interferences. In addition, a resolution manager that executes the codex rules at runtime is needed.

To summarize, we need a system, which is able to describe, detect and resolve interferences between Pervasive Computing applications dynamically. Based on our system model, we derive the following two additional requirements for this system:

First, the system must work on resource poor devices. As described in the system model, devices are highly heterogeneous concerning their resources. We cannot assume that we are able to access a resource-rich device in any given situation. Instead, the system must be prepared to work with a set of resource-poor devices only.

Secondly, the system must avoid distracting the user. Frequent interactions with the user could become annoying to the user and lead to a rejected system. Therefore, all steps in handling interferences should be automated as much as possible. As an example, whenever an interference occurs, the system should try to resolve it automatically. In some cases, automated resolution will not be possible. In such cases, the system must present the users with a description of the interference and allow the user to input a resolution strategy manually.

Clearly, these requirements conflict with each other. As an example, to automate the resolution process as much as possible, the system could monitor the user behavior and try to derive the right reaction to a given conflict automatically. However, this could lead to efficiency problems on resource-poor devices. In practice, we must find a suitable trade-off between our requirements.

5. Approach

Our approach for providing a Pervasive Computing system that allows the detection, description and resolution of interferences between applications builds upon our previous work in Pervasive Computing system software.[2][1][10]

We model a Pervasive Computing system as a number of *mobile ambients*. The ambient calculus was originally de-

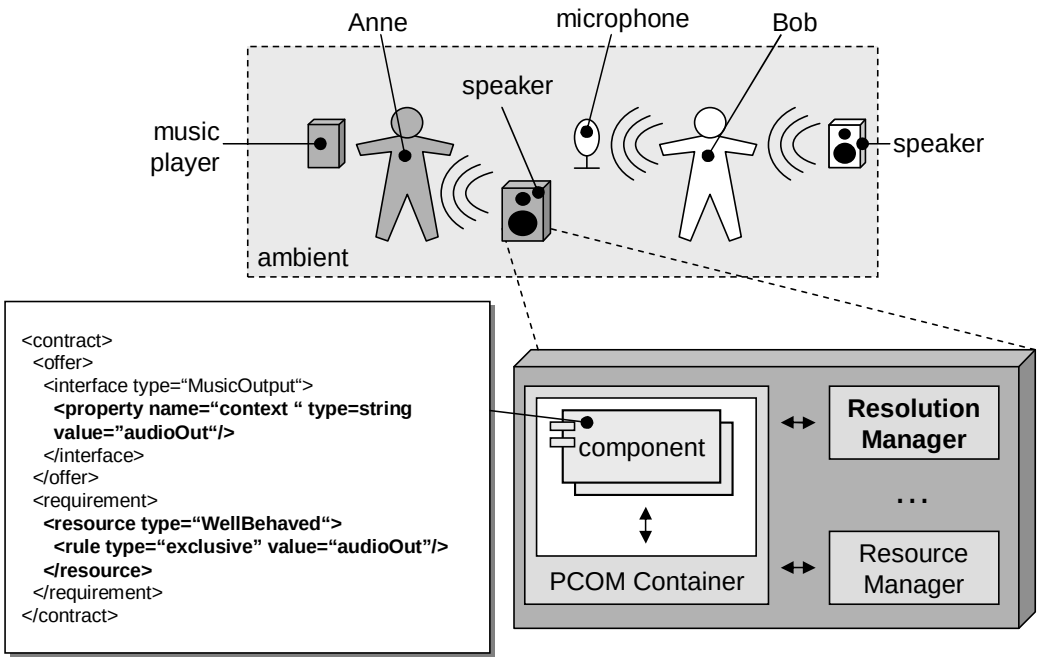

Figure 1. Extended system architecture

veloped by Luca Cardelli and Andrew D. Gordon [3]. As we have shown in [10], ambients can be used to model structured ubiquitous applications elegantly. An ambient denotes a bounded spatial area in which computation happens [10], e.g. a smart room, or a user's personal area network. Each ambient contains a number of applications, that are executed in its context.

Applications are modeled using our component system PCOM [1]. In PCOM, applications are composed from software components with contractually specified dependencies towards other components and their execution platform. As an example, a component could specify that it offers an audio output and requires a certain amount of memory and a speaker to do so.

To handle interferences, we introduce a new system component, the *resolution manager* into our system. In addition, we extend the PCOM component contracts. The resolution manager is responsible for detecting and resolving interferences. The contracts are extended to include all necessary information. An exemplary overview of the extended system architecture is given in Figure 1. New parts are written in bold.

In our extended system, each component specifies in its contract that it requires a new virtual resource, the so-called *WellBehaved* resource. Interferences are described by including specific requirements towards this resource. As an example, a component may add the requirement that no other component in the environment outputs audio. This is a natural extension of PCOM's contract model and allows the seamless integration of interferences into the component specification. To allow the system to detect interferences at runtime, application components specify in their offer contracts how they affect the physical context. As an example, the music application would specify that it outputs audio. To do so, we extend the PCOM contracts with new entries for context modifications. In the future, these entries could be derived and included into the contracts automatically. At the moment, we expect the component developer to manually specify the component's influence on the physical context.

In PCOM, resources are managed by so-called resource managers. Each resource manager is responsible for administrating a specific resource. At runtime, PCOM allows a resource manager to check if its resource is available and to allocate it to components as needed. If the resource becomes unavailable, the resource manager notifies the system, that one or more contracts are no longer fulfilled and that the application must be adapted.

To reuse this generic PCOM mechanism, we implement the resolution manager as a resource manager, which is responsible for administrating the newly added WellBehaved resource. This way, PCOM automatically delegates all information about the needed resource, i.e. our interference descriptions, to the resolution manager. If the resolution

manager detects that an interference occurs, it can access the current ambient, retrieve its codex rules and use them to derive a suitable strategy for resolving the conflict. If no other resolution can be found, the resolution manager can declare the component's contract as invalid and initiate the application to adapt itself.

As an example, if Bob's phone application defines in its contract that no other audio output is valid, Anne's music application could detect the audio output in the room but could not bind to it, as the PCOM resource manager would regard audio output as an already bound exclusive system resource. If Anne's application would already be running and Bob would enter the room, PCOM would detect, that it can no longer fulfill its contract towards Anne's music player and initiate its adaptation.

6. Related Work

We expect interferences between Pervasive Computing applications to be commonplace in future systems. Despite this, there is still relatively little work done in resolving such interferences automatically.

Morla and Davies presented a high level framework for describing interferences [4]. However, they do not provide an approach to resolve them.

Otto et al examine different strategies to resolve conflicts resulting from multiple users interacting with the same service at the same time [5]. Possible strategies are e.g. based on a first come first serve priority assignment or on explicit negotiation between users. Their findings provide valuable input for our work as the proposed strategies can be readily included in our system as exemplary codices.

7. Conclusion and Future Work

In this paper, we have presented a new approach towards handling interferences between Pervasive Computing applications automatically. Our approach is based on the ambient calculus to structure the interference determination. Codices are thus associated with ambients. A codex defines a set of rules that each application executed in the given ambient is expected to follow.

So far, we have defined our model for ambient based codices and begun to integrate it into our existing component system PCOM. At the moment we are analyzing different possibilities to describe interferences and the codices themselves. In addition, we plan to implement variants of the resolution manager both for resource rich and poor environments. A simple codex could resolve interferences by invalidating a component's contract, forcing the application to switch to another component. However, more complex resolution strategies are needed in practice. Such strategies must be designed and evaluated.

In the future, we want to examine the performance and usability of our system in a real system environment. As an example, we want to evaluate the overhead of our continuous interference checks. We also want to realize and evaluate a graphical design tool to allow end users to define new interferences and codex rules conveniently.

References

[1] C. Becker, M. Handte, and G. Schiele. PCOM – a component system for pervasive computing. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 04)*, March 2004.

[2] C. Becker, G. Schiele, H. Gubbels, and K. Rothermel. Base – a micro-broker-based middleware for pervasive computing. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom03)*, Mar. 2003.

[3] L. Cardelli and A. D. Gordon. Mobile ambients. In *Proceedings of the First International Conference on Foundations of Software Science and Computation Structure(FOSSACS98)*, pages 140–155, April/March 1998.

[4] R. Morla and N. Davies. A framework for describing interference in ubiquitous computing environments. In *Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops*, page 632, 2006.

[5] F. Otto, C. Shin, W. Woo, and A. Schmidt. A user survey on: How to deal with conflicts resulting from individual input devices in context-aware environments? In *Advances in Pervasive Computing 2006, Adjunct Proceedings of Pervasive 2006*, pages 65–68, May 2006.

[6] S. R. Ponnekanti, B. Johanson, E. Kiciman, and A. Fox. Portability, extensibility and robustness in iROS. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom03)*, 2003.

[7] M. Román, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, pages 74–83, Oct-Dec 2002.

[8] U. Saif, H. Pham, J. M. Paluska, J. Waterman, C. Terman, and S. Ward. A case for goal-oriented programming semantics. In *System Support for Ubiquitous Computing Workshop at the Fifth Annual Conference on Ubiquitous Computing (UbiComp03)*, 2003.

[9] S. Urbanski, M. Handte, G. Schiele, and K. Rothermel. Experience using processes for pervasive applications. In *Pervasive University Workshop at Informatik 2006*, 2006.

[10] T. Weis, C. Becker, and A. Brändle. Towards a programming paradigm for pervasive applications based on the ambient calculus. In *Proceedings of the International Workshop on Combining Theory and Systems Building in Pervasive Computing (CTSB), Pervasive*, 2006.