Active Surroundings: A Group-Aware Middleware for Embedded Application Systems

Dongman Lee
Information & Communications University
Yusung-Ku, Taejon, 305-714, Korea
dlee@icu.ac.kr

Abstract
In this paper, we present Active Surroundings, a group-aware middleware infrastructure for embedded application systems where entities (devices or services) actively respond to user actions and help users to perform their jobs with no or minimal involvement of users. Our system focuses on two key issues: transparent application reconfiguration and group-context awareness.

1. Introduction
With the ubiquity of networks and computing devices, pervasive computing [5] where applications are being embedded into our environment is expected to become reality in near future. For this, the embedded application system is required to anticipate requests from users, which is referred to as context-awareness, and has to reconfigure the required service with services available in the new context where the user moves and provide it to him transparently, so called dynamic application reconfiguration [6]. Several software infrastructures [1, 2, 4, 6] have been proposed for embedded application systems.

In a real world, users usually interact with others when they perform their task. During interactions, their intentions may conflict with each other and it is difficult to pre-define all the conflict situations since interactions among people are dynamic. However, the existing systems either provide no support of the group context or require applications to specify the group context explicitly. In these systems, binding the required services in a new environment to a user application has to be done either in a pre-defined way or diminished to only available services when appropriate services are not available in the environment. We present Active Surroundings, a middleware infrastructure for embedded application systems. The proposed system aims to support a group-aware embedded application system where the system minimizes user involvements when users move from one environment to another.

2. Key Considerations

2.1 Transparent Application Reconfiguration
Embedded Application systems aim to allow nomadic users to access and manipulate information anywhere and anytime without directly dealing with underlying system details. For this, we need to deal with two issues: service transparency and flexible adaptability.

- Service (Application) Transparency
  It is common that an application should be “manually” reconfigured whenever it is introduced to a new domain. However, it is difficult for an application to adapt itself to dynamically changing environments because application developers cannot design it to cope with all changes required by the environments. The system should support transparent reconfiguration for both users and application developers with minimal administration.

- Flexible adaptability
  Dynamic nature of embedded application systems makes us impossible to assume that exactly the same services are available in every environment. The system is required to find and adapt a semantically similar service or resort to the user’s intervention if the required type of a service does not exist.

2.2 Group Context Awareness
Context-awareness enables an embedded application system to adapt to users proactively without distracting them. Context-aware applications interpret the contexts collected from various sources, and then, deliver services or adapt themselves to what a user wants to make them in given contexts in advance. However, since the existing context-aware applications are totally independent of each other, it may intrude others’ interest by accident in a group-aware environment where conflicts among them frequently occur. Thus we need to
not only consider the contexts of individual user but also put them together into a group-context, a set of context of individual users. In fact, a group context is more than the context of a collection of individual users. We take into account contexts of individual users as group-contexts in terms of conflicts among them, harmonization, and a new context comes out only when users are together as a group. At first, we focus on the conflict of contexts because it is a critical issue in a shared environment where a set of personalized context-aware applications run for different purposes independently.

3. Proposed Architecture

3.1 Transparent Reconfiguration Support

For service transparency, we introduce a Service Link object. It provides a common interface for registering adaptation mechanisms or system services which enable transparent service reconfiguration. A service link object is created per a service object and keeps service information using reflection. It also acts as a proxy for a local communication broker. When a service object moves to a new environment, the link object of a target service object is rebound to the communication broker of the new environment without direct involvement of the service object.

To support flexible and dynamic adaptability, we leverage the notion of behavior sub-typing [3]. To relax the constraints, we allow service providers to use either the strong or weak notion of the behavior sub-typing, which tells if an additional operation of the sub-type can be represented by a combination of super-type’s operations or not. Therefore, the scheme substitutes the required service with either its sub-type or its super-type in run-time when it is not available at the target context. In our scheme, the target application can be flexibly rebound with the same type of locally available services and functional correctness is guaranteed.

3.2 Group-Context Management

In a group-aware environment, the context management coordinates context-aware applications so that they do not sacrifice others unintentionally to achieve their goals. To deal with conflicts of context-aware applications, we have defined conflict detection and resolution models. To dynamically detect a conflicting situation which cannot be specified at the time of the application design, we build a service ontology which classifies the actions of services into the effects – increase and decrease in our cases – on contexts and the states of services. For example, A Dimming service performs two actions, StartLampToLevel and StopLamp. The first action increases the value of the context, ‘Brightness’ but the second action decreases it as depicted in Figure 1. A conflict is detected when two contradictory actions are performed on a context by two different users’ context-aware applications. By using the abstracted semantics of actions of services, we can detect conflicts without pre-described descriptions of conflicting situations.

![Figure 1. Conflict Detection Model](image)

The conflict detected at runtime, is resolved based on the preference of users. The preference can be given by users or application programmers directly or learned from users’ behaviors. We use a compromise action between two conflicting actions by using the weighted sum of arguments of an action and a user preference as a resolution policy of the conflict.

4. References