Towards a Software Architecture for Device Management in Instrumented Environments

Christoph Endres
Saarland University
Saarbrücken, Germany
endres@cs.uni-sb.de

ABSTRACT
An infrastructure for scalable plug-and-play device management in an instrumented environment is presented. A prototype of the system is described and issues of the overall architecture are addressed.

1. INTRODUCTION

The FLUIDUM project (www.fluidum.org) is currently building an instrumented environment in order to investigate interaction techniques for ubiquitous computing. An infrastructure for device communication has to be provided that allows fast prototyping and provides a stable foundation for the projects devices, scalable to desk-, room-, and building-level.
In analogy to the well known concept of a window and driver manager of a conventional desktop computer I am working on such an infrastructure for instrumented environments of various scales.
At the core of this system is a device manager with a dynamic plug and play mechanism for possibly fluctuating devices (e.g. PDAs or laptops) in the environment. A prototype of this device manager is built. Its architecture is described in the following section. Finally, I discuss issues raised during the implementation of the prototype.

2. ARCHITECTURE OF THE PROTOTYPE

2.1 Design goals

The design of the device manager is guided by several constraints. In the FLUIDUM project it will be used in three differently scaled instrumented environments, with a potentially widely varying number of devices and applications. Also, in order to cooperate with other, similar projects at the same office, the device manager has to be reusealbe in other contexts. In order to achieve these goals, there are several important considerations.
Since the architecture has to be open to new applications and new devices, the interfaces have to be well defined and simple. The architecture has to be sufficiently flexible for unforseen future devices. This will be achieved by the way devices are classified, as described in more detail below.

2.2 Overall system design

The core part of our system is a blackboard, called "the pool". It is used to store and exchange all sorts of important information about the environment. Connected to the pool are several services that provide information for the pool, or offer processing of information and then eventually write their results back to the pool.
The plugboard service is one of those services and deals with the device management. Besides keeping track of all plugged devices and their features, it can trigger actions on the devices based on data stored in the pool. For instance a service might put a request for taking a photo with a digital camera on the pool. The plugboard service then takes the request from the pool, captures the requested photo and places a reference (URL) to this photo back in the pool. The requesting service can take this URL and process it.
The next section discusses the device manager's approach for device classification. After that, the architecture of the plugboard is presented.

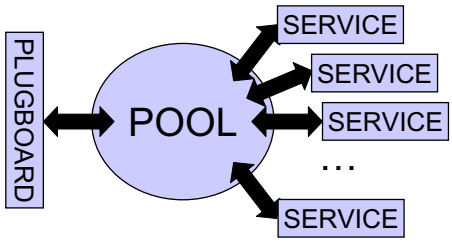

Figure 1: High level view of the system

2.3 Classification of devices

As mentioned above, one main issue in device classification is the uncertainty about future devices. At the current pace of hardware evolvement, it is very hard to tell which kind of devices will have to be integrated in the system in a few years, and next to impossible to find a classification of devices that could handle them. Therefore, we decided not to classify the devices, but instead to classify the different properties of a device (video capturing, text entering, infrared sensing, etc.) and model a device as a list of those properties.
This approach turned out to be very flexible and useful so far.

2.4 Plugboard architecture and device manager

The architecture of the plugboard reflects the approach of device classification. A device is modelled as an object containing a list of parameter/value pairs (e.g. "name=camera01") and a list of property APIs. The inclusion of such a property API, e.g. "video in", means that the device has this property. If a property of this type is missing, we can assume that the device can not perform that task. The advantage of modelling those properties as API is that besides getting information about the device, we also acquire access to its features. The APIs are standardized, so on encountering a certain property API we know which functions can be called.
The central part of the plugboard is the device manager server. It is a lookup service to which devices can connect or from which they can disconnect. On the other hand, services can also connect to the server and request information about devices. Each connected service will be automatically informed if there are important changes in the plugged devices. Some of those services take care of the connection and exchange of data to the central data pool.

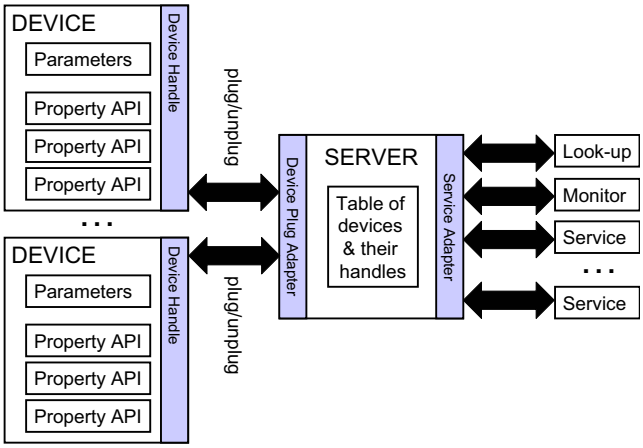

Figure 2: Architecture of the plugboard

3. DISCUSSION ISSUES

There are some unresolved issues in the current system that I would like to discuss.

3.1 Centralized design as bottleneck

Although the central device manager service seems the logical design approach, it is a potential bottleneck and source of instability of the whole system if it fails. At the moment, the services keep a copy of the device manager's list of all plugged devices and thus could continue working during a device manager failure. Although stable, this solution might lead to performance issues. Alterantive approaches might include self-organizing structures or some sort of peer-to-peer network.

3.2 Reliable recognition of device disconnection

At the moment, the devices connect via remote method invocation to the central server. Although there are some stable mechanisms to detect a failure of this connection, there is no sophisticated mechanism yet to detect failure of a device without previous disconnecting.

3.3 Ressource management

The device manager server is a useful lookup service to find available devices and to find out about their features. A feature and concept for scheduling devices to applications is yet missing. Especially a realiable locking mechanism for devices or device features in use is missing. Also, mutual locking of different properties on the same device is missing. For instance, a camera currently in use in the system is not capable of simultaneously broadcasting a video stream and capturing a high resolution photo. Those dependencies have to be modelled.

3.4 Inclusion of future devices

This is a point which should be solved with our approach of device properties. The author would like to discuss it and gather some more opinions.

3.5 Dealing with virtual devices

Some properties, for instance recognition of visual markers, do not have a hardware equivalent but are overlays of other properties, for instance video capturing in this example. There current solution is implementing virtual devices that plug to the server both as device (marker recognizer) as well as requesting service (looking up devices with video capturing property). Although this approach works, there might be a more elegant way to do this.

4. ACKNOWLEDGMENTS

This work has been funded by the German Research Council (DFG) and the Chair for AI at the University of Saarbrücken, Germany.