Middleware for sensor networks, and a context-aware music player

Andreas Hallberg
KTH Royal Institute of Technology
ID2012 Ubiquitous Computing
anhallbe@kth.se

I. Introduction

This report presents my project for ID2012. It consists of two parts: **Sense** - middleware that allows various types of sensors to communicate in a very decoupled way, and **SenseMusic** - an application that lets the user play music without having to care about the specific application/device to use for output.

II. Sense

The idea behind the middleware is to allow sensors to communicate over the internet, and for context-providers (which is basically just another sensor) and applications to use the data for ubiquitous computing [3]. The goal is to make it easy to add/remove sensors and applications without having to reconfigure any other parts of the system.

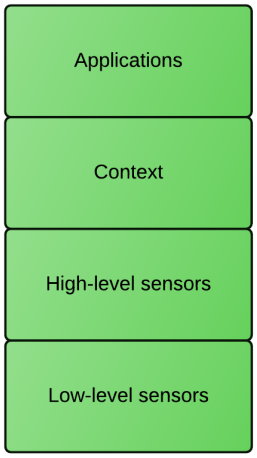

Fig. 1. Sense Stack/Abstractions

Figure 1 shows the basic abstractions used. A **low-level sensor** can be any sensor that senses the physical world and outputs low-level values, e.g GPS coordinates, temperature, signal strength etc. A **high-level sensor** is one that read values from other sensors, and outputs some abstracted value. For instance a sensor that reads values from all temperature sensors in a home, and outputs the average. This makes it easier to combine multiple sensors (for accuracy and/or redundancy) and some logic to build a **context**. The context is essentially just another high-level sensor, but it should be a bit more specific about *what* a user is doing and *why*. Ideally, an **application** should only be interested in contexts, so they can be closely coupled. Applications use the context to produce some output back to the physical world. One example of an application is **SenseMusic**, which is described later in this report.

A. Architecture

The system (Figure 2) is essentially a centralized publish-subscribe service, which is accessed either with a Java client (**jSense**) or through a REST API. Sensors can publish value updates, and higher-level sensors, context providers and applications subscribe to *search queries* that match their interests. There is no direct communication between clients, which makes it easier to add/remove them. The centralized part (**SenseService**) is hosted in a public cloud, which provides scalability and fault-tolerance.

The SenseService consists of two parts; a NodeJS web server and Elasticsearch [1], a distributed search engine. Both of these components can easily be distributed among physical/virtual machines, which should deal with most of the drawbacks of using a centralized system (single point of failure, scalability, redundancy etc).

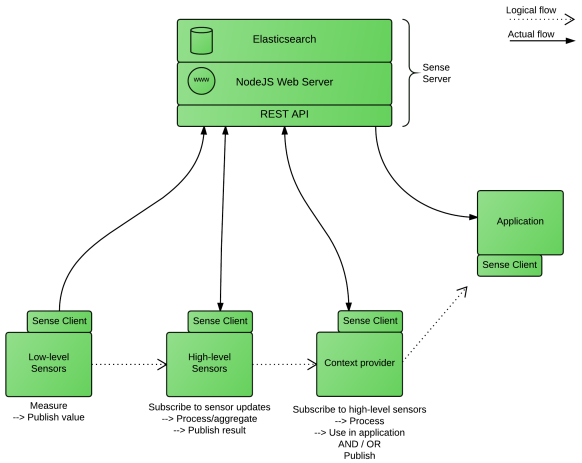

Fig. 2. System architecture and data flow.

B. Using the middleware

Sensor updates/publications consist of four parts:

- **name** - Identifies the sensor.

- **description** - This should be an in-depth description of the sensor's purpose, position etc. Anything that will help others to automatically find it with a search query.
- **valueType** - An enum that will make it easier to parse the value, e.g string, integer, geoloc, float.
- **value** - The new value of this sensor.

This format is used by all sensors (low-level, high-level, context). An update can be published with the java client (jSense):

```
sense.publish(new SensorUpdate(
   "PhoneWifi",
   "SSID that my phone is connected to",
   "string",
   "eduroam"
));
```

If somebody is interested in updates from the above sensor, they could subscribe to the following:

```
sense.subscribe(
   "(wifi OR ssid) AND phone",
   new UpdateListener() {
      public void onUpdate(...) {
          //callback
      }
   }
);
```

III. SenseMusic

This is a context-aware music player that uses the Sense middleware [4]. The idea is to allow the user to listen to music on a nearby device, without to interact with that specific device/application. Playback is controlled by shaking an Android-phone. By shaking the phone, the system should automatically detect the best device for music playback and start a suitable song.

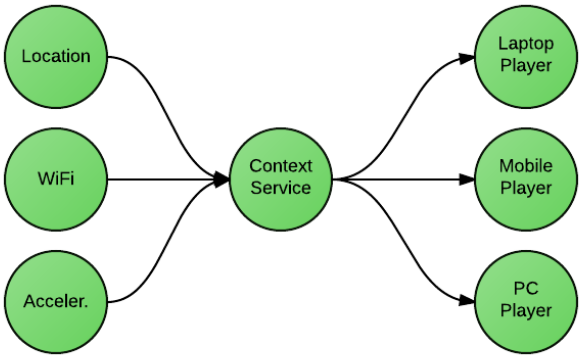

Fig. 3. Components of SenseMusic. From left-to right: low-level sensors, high-level/context, applications. The context service and the music players are specifically developped for this application.

A. Low-level sensors

There are currently three low-level sensors in the system, all of which are located in the users's Android phone: Position (GPS and Network), WiFi SSID detector, and an accelerometer. The Android-application Tasker [2] is configured to send updates to the Sense service when certain conditions are met. The following low-level events are posted with appropriate descriptions, values, types etc:

- PhoneWifiConnect
- PhoneWifiDisconnect
- PhoneLocation
- PhoneShake

B. High-level sensing and context

A separate service is listening for the events in order to figure out what is going on and what the user wants to do. It uses the Wifi- and Location events to decide whether the user is at home, in school, or mobile. When a PhoneShake event is detected it understands where the user is, what it is doing (to some degree) and that it wants to play some music. It will output one of the following events:

- "Play music on laptop"
- "Play music on PC at home"
- "Play music on phone"

The context is basically decided by checking the state of the "WiFi"-sensor and the "location"-sensor. For example, if the location value is within 50 meters (arbitrary range) from Electrum **or** if the WiFi SSID value is "Eduroam", then the user is probably studying and wants to listen to music on "laptop". A similar strategy is used to decide that the user wants to listen on "PC at home". If the "location" value is far from home and Electrum, **and** the current WiFi SSID is not "eduroam" or my home-network, then the user is probably mobile and wants to listen to music on "phone". The ability to account for the age of a sensor value is only partially implemented at the moment.

C. The music application

This application runs on each device capable of playing music, i.e a laptop, a PC and a phone. Whenever a "Play music on X" event is detected, it will start playing a randomly selected song to the default output device (embedded speaker, headset, plugged-in speakers etc.).

IV. Future work

There are a few parts of the system that can be improved:

- **Push publications** - At the moment, the Sense middleware periodically polls the central service for update events. My intention was for the service to push these events (for instance through websockets) but that isn't implemented yet.
- **Sensor distinction** - There is no way to distinguish low-level, high-level and context sensors aside from

naming conventions. A more structured way to do it would be to add a new ”sensor level” field.

- **Sensor identification** - Sensors are uniquely identified through an ID given by Elasticsearch. This could probably use some abstraction and it would be better to use the sensor name to identify it.
- **API transparency** - The Sense API exposes some functionality that should probably be abstracted, such as the Sense Service hostname and polling interval.
- **SenseMusic sensors** - SenseMusic only takes two simple sensors into account to build a context (WiFi SSID and location). This is enough for the scope of this project, but in practice you would need to add more dimensions, e.g the user's calendar, usage patterns, music taste, and of course more physical devices.

V. Conclusion

I've developed a general-purpose middleware for sensor networks that works and is fairly simple to use. It is used by an application that plays music based on the context of the user. The application is very simple and there is definitely room for improvement, for instance by adding more sensors to the system.

I have successfully used the middleware for another project not related to this course, which showed me that other people can use it by just downloading the software and reading the documentation [5].

References

[1] *Elasticsearch: RESTful, Distributed Search and Analytics*, https://www.elastic.co/products/elasticsearch

[2] *Tasker for Android*, http://tasker.dinglisch.net/

[3] *Sense on GitHub*, http://anhallbe.github.io/sense/

[4] *Sense Music on GitHub*, https://github.com/anhallbe/sense-music

[5] *”KnockKnock” Ericsson E-hack project*, A. Hallberg, O. Lundh, A. Zaitov, https://github.com/anhallbe/ehack