Context-Based Service For Intelligent Public Transportation Systems

Valeriu-Daniel Stanciu, Ciprian Dobre, Valentin Cristea
University Politehnica of Bucharest
Bucharest, Romania
valeriu.stanciu@cti.pub.ro, ciprian.dobre@cs.pub.ro, valentin.cristea@cs.pub.ro

***Abstract*—Nowadays, cities become ever more populated, starting to represent an enormous source of pollution, congestion and overcrowding. Technology is evolving at a very fast speed; most of the people have a smartphone and are using it on a daily basis. In this paper we describe a service designed to assist citizens take informed decisions regarding the means of transportation, to make them more aware of public transportation alternatives. We propose techniques to deal with mobility of people (monitor individual and group traveling conditions), and propose alternative means of transportation considering the active use of context information. The idea is to use the mobile apps capable of understanding the users environment and reacting accordingly.**

***Keywords*-context-awareness; smartphones; intelligent transport systems; mobile computing;**

I. Introduction

Urban mobility is an attendant of modern city life and its sustainability heavily depends on the consumption of different resources, such as electricity and fuel; therefore, it manifests in carbon dioxide (CO_2) and local pollutant emission and directly influences the quality of life. Urban traffic is responsible for 40% of CO_2 emissions and 70% of emissions of other pollutants arising from road transport. Road transport is identified as being responsible for more than 70% of the GHG emissions from the entire transport sector and for around 13% of total emissions of all sectors (transport sector represents almost 20% of total emissions). Intelligent Transportation Systems (ITS) aim to overcome these problems by introducing solutions where specialized infrastructure, communication technologies and innovative services are used to enhance the quality of traffic and mobility management, multi-modal or road transport solutions.

The need for fast and intelligent transportation is, thus, of great demand. People must be able to get easy and safe from one part of the city to another, but the lack of information may cause a lot of problems. Most importantly, the lack of timetables for bus, tram stops, or the bus routes hinders the decision on the most suitable alternative for people living inside the city. The public transport sector should be well connected within the intelligent city - and the solution is at hand. Smartphones and tabletPCs are today ubiquitous mobile devices, and they provide rich processing capabilities, solid technical assets and real-time connectivity at the tip of the finger.

In this paper we present an ITS service (and a mobile-side app) designed to assist citizens take informed decisions regarding the means of transportation, to make them more aware of public transportation alternatives. We propose techniques to deal with mobility of people (monitor individual and group traveling conditions), and propose alternative means of transportation considering the active use of context information. The idea is to use the mobile apps capable of understanding the users environment and reacting accordingly. These are the so-called "context-aware mobile applications". They are designed to make automated decisions, such as mute the phone when user is in a conference, recommend events based on users profile, or display relevant information about public transportation network.

The mobile app uses sensors (i.e., Wi-Fi and GPS signals, etc.) to determine users actions and determine whether the user is waiting for the bus, or leaving home. It then sends him relevant information, such as distance to the closest bus and tram stops, lines passing nearby, timetables of those lines, full routes of the lines (all the stops from their routes) and many other useful pieces of advice.

The rest of the paper is structured as follows. Section II discusses related work. Sections III and IV present the proposed solution, from methods and techniques to the proposed architecture. Section V describes the details behind the real-world implementation. Section VI presents experimental results and analyzes the obtained performance. Section VII concludes and presents future work.

II. Related Work

The proposed system includes both a data acquisition layer, in charge with real-time sensors management, and the context layer, providing analysis of situation and feedback of relevant information. Given the rich functionality provided by modern smartphone, today more and more authors propose methods and techniques to deal with context adaptation and mobility aspects.

CARISMA [7] is a mobile computing middleware that provides adaptive and context-aware services for applications, exploiting the principle of reflection. It offers applications an abstraction of the middleware as a dynamically customizable service provider, where customization takes place by means of metadata. It encapsulates the context data of an application into an application profile, which specifies

associations between the services, the policies applied in order to deliver those services and the context configurations. An application can modify its own profile in the middleware using a reflective API. Similarly, CARMEN [6] provides context-aware resource management for automatic reconfiguration of wireless Internet services in response to context changes. Each user has a proxy which provides access to the needed resources. If a migration occurs, the proxy makes sure that the user still has access to his resources in the new environment. The context is represented by two types of metadata stored in the middleware, profiles and policies. Access rules per profiles, as well as binding and migration information, are specified in the policies.

Google Now [2] works as an intelligent personal assistant, providing the user real-time relevant information based on his previously-learned habits. Google Now predicts users behavior and displays relevant information in the form of "cards". Specialized cards currently include the following: birthdays, events, flights, movies, places, sports, traffic, weather, etc. Unlike such frameworks, who are more generic, we target the use of context-based adaptation for information augmentation in smart city scenarios. The context model and means are, as we detail later, different between these cases.

In terms of public transportation, there are very scarce applications that can be considered truly intelligent. Transport Urban [4] is an active project in this area. The application is a basic route planner for public transportation means. It takes current location and destination as inputs and it provides the means of transport you have to use in order to accomplish your route. Although a useful application, it does not imply context-awareness – the solely task is to compute a route from manually inserted inputs.

III. Architecture

A. Architecture Design

The architecture consists of three logical layers (see Figure 1), each one accomplishing a particular task: 1) Sensor Data Collection Layer; 2) Context Processing & Situation Identifying Layer; and 3) GUI & User Interaction Layer.

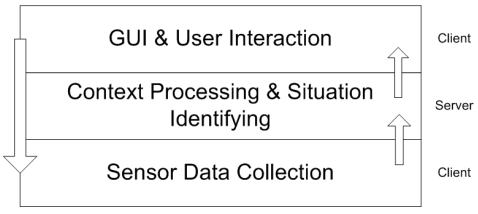

Figure 1. The Proposed Architecture.

The implementation of the specified architecture (see Figure 2) is based on the client server paradigm. The mobile-side app is installed on the smartphone. The first layer of the application deals with monitoring services, which continuously collect raw data from sensors (e.g., GPS location, battery level). This is part of the client-side processing.

The second layer deals with processing data gathered from the clients sensors, and deriving situations based on predefined rules. These tasks run in a server environment, mainly because they involve higher computational capabilities.

Finally, the third layer is responsible with the visualization and user interaction. It runs on the users smartphone (client) and makes decisions based on messages provided by the server. We include here the possibility of click-and-action (a basic user interface providing implemented actions).

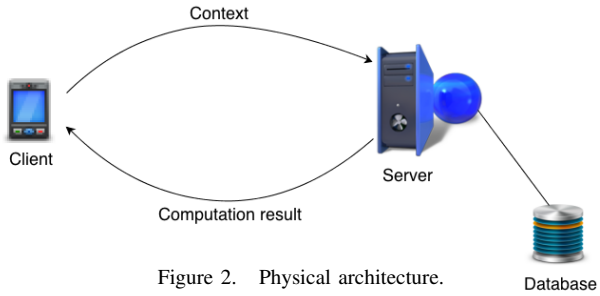

Figure 2. Physical architecture.

Following the client-server paradigm, the mobile app sends messages with contexts, and the server responds with messages containing the result. On the server-side, a database is populated with information about the public transportation network (lines, routes, timetables, geographical positions, etc.).

B. The Context Model

Context represents any information used to characterize the situation of an entity involved in the user application interaction [8]. Here, we define the entity as being any person, place or object which is relevant to the interaction between a user and the application, including the user and the application. Even if for the pilot demonstrator we currently use mostly location-related data (e.g., GPS coordinates), we aim to extend this with more information, like speed (accelerometer), users schedule (agenda), WI-FI info, current time, etc. A context model is used to link together information provided by the smartphone, to develop increasingly intelligent features.

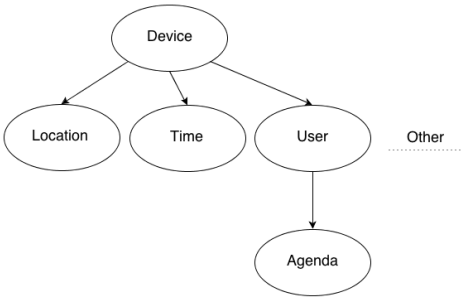

Figure 3. Context Model being used.

Currently, the most important context-related information is location - obtained from several sources (i.e., GPS and

GSM capabilities on the device). A module periodically collects information, and interprets and encapsulates the raw data into a recognizable form of location data (latitude and longitude). However, we make the comment that the mobile app follows a modular design, so new modules for collecting data can be easily written and added as new functionalities are implemented in the future.

IV. Operation modes

The system considers two scenarios, depending on the situation the user is in:

- User is in a public transport station and s/he is waiting for a means of transport
- User leaves home

We started the design from real-world problems we all face every day. In the first scenario, the user is located in the perimeter of a public transport station, and s/he remains there for a period of time waiting for a means of transport (bus, tram, etc.). What would anyone want to know while waiting for a bus? How long is he going to wait? What bus should he take? Where would that bus take him? The developed mobile app responds to exactly such questions before being asked, offering to the user the necessary information.

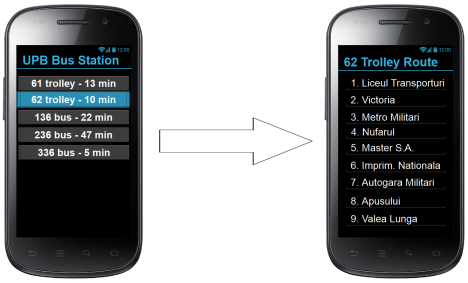

Figure 4. Situation 1. User is waiting for the bus.

When the first situation is identified, the smartphone buzzes to notify the user. When the user opens the screen, s/he sees the name of the station (UPB in Figure 4), the alternative means of transport passing by there (the 61 and 62 trolley, and buses 136, 236 and 336), as well as their timetable (succession intervals). If the user selects one of the lines (e.g. 62) s/he will be able to see its entire route.

Another interesting situation we considered is when the user leaves home - s/he starts receiving GPS signal, and it's the first time we localize him/her. If the user starts using the mobile app, we assume he wants relevant information.

In this situation, when the user opens the screen, s/he sees all the transportart pick-up stations around his area, along with corresponding distances (UPB - $100m$, Bd. I. Maniu $500m$), lines passing through them and their timetables. If the user selects one of the lines (e.g. 35) he will be able to see its whole route, exactly like in the first situation (see Figure 5).

Figure 5. Situation 2. User is walking.

V. Implementation details for a real-world pilot service

We considered a pilot service implementation to support the public transportation infrastructure in Bucharest, Romania. First, RATB [3] is the public ground transportation operator in Bucharest. Administered by the City Municipality, it operates a complex network of buses, trams and trolleybuses, covering the city, together with Metrorex, the company in charge with the underground transportation (the subway).

The current starting point is as follows. Bus and tram stations have signs with their names and lines passing through them, with few services providing info about lines (i.e. info-points, interactive screens). Even if other Europe cities are more advanced, they fail to provide complete service support (such as what alternative means of transporation to choose, such that to minimize the traveling time and increase the chances of succesive connections). For Bucharest, *ratb.ro* is RATBs official website, and provides minimal information, with little integration with the transportation infrastructure.

The first idea towards developing the operation modes previously presented, was to make the user aware of the precise moment when the next means of transport arrives. RATBs newest buses have GPS systems integrated with an in-house application, which shows relevant information on a display located inside the bus (including the next station and current GPS position). However, for security reasons the company cannot grant access to its central management platform - so this approach is not fit at all for our application. We next considered estimating the time of arrival – given the succession intervals displayed on the website, we can try calculate an approximate waiting time – but, naturally, the traffic in Bucharest (as with any other city) is very unpredictable, making the task almost impossible.

Thus, the feasible solution was to deliver the raw content displayed on the official RATB website (line info, succeeding intervals, stations). Other approaches would have made our application quite inaccurate, nearly unusable, rather ridiculous and avoidable. Surprisingly, in our experiments the users where quite happy with simpler info.

A. Client-Side - the smartphone app

On the mobile side, the app is using the smartphones sensors and some of its computing capabilities to capture

relevant context information. For our needs, we use mostly data from the GPS sensor, along with Wi-Fi and mobile Internet (for online communication).

For ease-of-development, the pilot implementation was done over Googles popular Android platform. The mobile app includes several screens:

- *RunningActivity* the first screen, which checks the execution requirements (i.e., it can display a message prompting the user to activate the GPS service if needed);
- *StationsWithLines* the screen is activated by a message containing relevant information, received from the server; it shows a list with stations and lines passing through them;
- *LineWithStations* shows a list of stations corresponding to a public transport line;
- *MapStation* shows the current position and the stations position on a map.

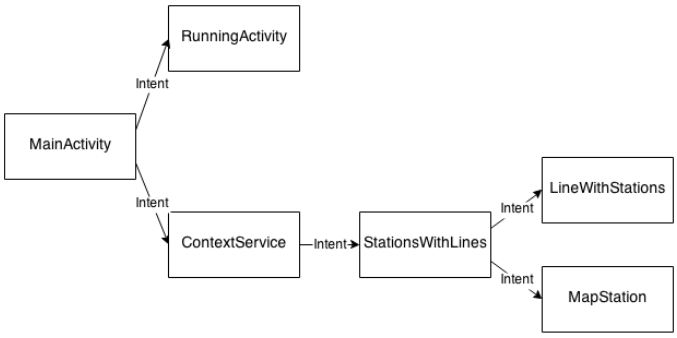

Figure 6. Component Diagram.

RunningActivity displays general information about the application. The main purpose is to represent a basis when the application is idle or the user does not fit with any of the covered situations.

StationsWithLines is shown when the user is detected to be in one of the two situations. *LineWithStations* and *MapStation* are the next views. The former is started when a public transportation line is selected from the list, displaying all the stations the line has, while the latter is opened when a "Map" button is pressed for one of the surrounding stations, displaying current position and station position on the map.

The mobile app deals with several modeled entities. Conceptually, the mobile app needs to display first a *Station*, which represents a public transportation station. This includes a name, a street location, the distance from current location, latitude and longitude positions. But public transportation vehicles run through the stations, there are many different lines, so we also display a *Line*. A line has a number, and a type (i.e., bus, trolley, tram). The user also needs to see an *Interval* (a line has many succeeding intervals, depending on time, day of the week, etc.). The interval contains information about the day of the week, succeeding intervals (i.e., between $6:00AM$ and $9:00AM$, $9:00AM$ and $13:00PM$, $13:00PM$ and $18:00PM$, and so on.

The mobile app also tracks users movement. It provides a *Location* type parameter, which specifies the current location detected by the GPS system. We memorize two consecutive locations, *exLocation* and *currentLocation*, as a basis for context processing and situation identifying. After the current location is obtained and the *exLocation* is updated, the client-side application invokes a method to send a request to the server for an information update.

For the communication layer, we opted for the use of REST-like (Representational State Transfer) services. The amount of data sent over the network is significantly smaller than using alternatives (e.g., SOAP). Context information is fed back to the server using a JSON format (language-independent, with parsers available for many languages).

The response consists of a JSON-encoded array of Stations (only one station if the user is waiting for a means of transport). Given the response, we populate the view with stations, together with corresponding information: name, street, latitude, longitude. Figure 7 shows the *StationsWithLines* views when given a UPB location as a first location. As seen, the user is presented with data about parents (stations names with streets and distances) and children (second view lines numbers, with types and succeeding intervals).

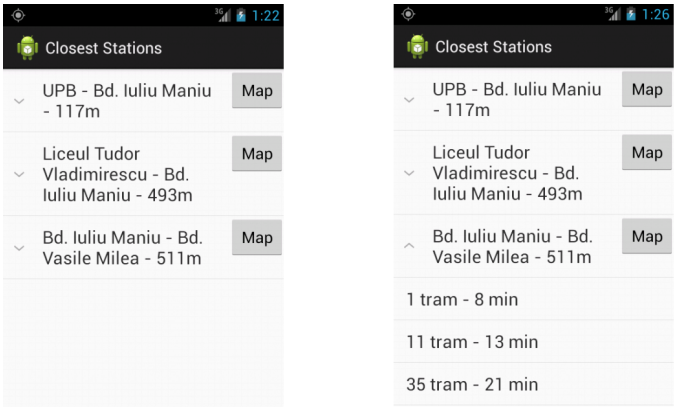

Figure 7. Closest stations view, unexpanded (1) and expanded (2).

A similar view is displayed when the user waits in a public transportation station. The only difference is that there is only a single station, with 0 meters distance.

When a public transportation line is pressed, *LineWithStations* is started.

When the "Map" button is pressed, the user is presented with *MapStation*. Here, we retrieve the map and place two markers on it, representing the current location and the location of the selected station (see Figure 8(a)).

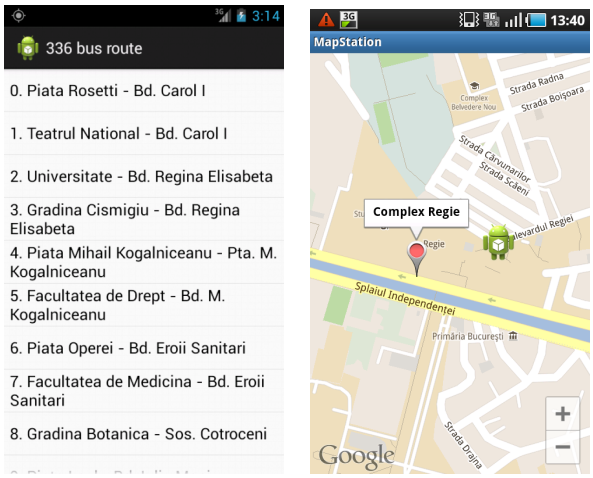

Figure 8. (a) LineWithStations view. (b) MapStation View.

B. Server-Side

On the server-side, every decision was motivated by the needs - the database has to manage large sets of data, and support complex queries. The server is currently implemented in CakePHP [1], an open-source web application framework written in PHP, which follows the MVC software design pattern.

In terms of entity modeling, we have delimited: *Station*, *Line* and *Interval*. A station has a list of lines, a line has a list of intervals, and a list of stations. The first thing the server does is to decode a message received from the mobile app. *processLocationUpdate* is called when the server sends two locations within a message. First, we retrieve the current and ex latitudes and longitudes. Then, we detect whether the user is waiting or not, by verifying the differences between current and previous position. If the user does not move, we search for a station corresponding to the location s/he is waiting (the user can wait for something else, not necessarily a means of transport). We return the station (if it exists) along with lines and all necessary information.

getNearbyStations is somewhat similar (the search area is a little bit wider). The main difference is that here we calculate the distances between the user and his/her closest stations, and add them to the response. To compute the distance we use the *haversine formula*, an important equation used in navigation.

VI. Experimental Results

Functional testing has been intensively done, hand in hand with the development process. Screenshots obtained were previously presented. We have also run field tests to see how the application works in real-life conditions. The experiments were conducted on a Samsung Galaxy S GT-I9000 smartphone.

It is known that the GPS drains a lot of power. We had to make a decision regarding the time interval between GPS updates. Anyone would prefer to have real-time positioning, but this would burn down the battery really fast. Thus, there were two main problems we had to overcome:

- The time between GPS updates had to be short enough to detect the moment when the user starts waiting and react quickly providing relevant information;
- The time had to be long enough not to drain the battery too quickly; another important thing was that the user had enough time to definitely walk more than the distance a public transportation station covers.

30 seconds was, in the end, a good observation interval. A shorter interval quickly leads to false positives (users detected as waiting for a bus, even if they were just passing through a public transportation station), not to mention that the battery damage would have been huge. A longer interval would make the mobile app useless, information being received too late, and users becoming rather frustrated.

We conducted several battery tests for a period of observations of 1 week (experimented different situations over 5 different smartphones). A relevant metric is the ratio between the situation when our application was running and the other considered situations (see below), ratio which was rather constant across different smartphones. An example of obtained results within an hour on our own test smartphone is presented in the following table:

Situation	Battery Consumption
Google Maps app running	30%
Our app running	8%
Standby	6%

We represented three situations: the user accessing a maps application (same popular app was used in all experiments), our mobile app running, and finally the situation where the smartphone is not used at all (normal standby). As seen, our application battery consumption is very low, only 2% over the standby battery consumption. Another observation is that, within an hour, our application had, in average, 46 seconds CPU usage, and a median of 2% battery share (92% was for the screen, and 6% for the GSM cell service).

The second resource in discussion is the Internet connection - another costly resource. GSM providers give limited traffic to the users, and we do not want our applications users to pay a high price for the Internet traffic because of the app. Thus, we carefully considered what and when to send (in terms of data traffic).

The mobile app uploads and downloads data from the server. The size of a sent package is almost always the same, implying a fixed size of traffic upload per request. The only variation is when the user first exists the building (that package is smaller because is does not include an ex location).

Most traffic is consumed by the received packages, but we send only the most relevant information. For example, when no relevant information is found on the server, we send an empty response.

We made several network traffic tests too. They revealed the following figures:

- A location update sent to the server is worth $150B$ upload;
- An empty response is worth $50B$ download;
- A full response (in case the user is located within a station) is worth an average of $10KB$

When the user receives valid information from the server, s/he stops contacting the server until is detected as leaving the station. We do this to save traffic generated by redundant information. For example, lets say a user enters a public transportation station and stays there for a while, waiting for the bus. At every 30 seconds, more than $10KB$ are consumed for retrieving the same information. This is not too efficient. For 10 minutes, s/he will consume a minimum of $200KB$ of Internet traffic, for every station hell wait the bus in. This is conforming to osbservations about the importance of background traffic, as notices by other authors too [5]. A commuter usually waits for means of transport for an average of four times a day. This means $800KB$ per day, $24MB$ of redundant data per month.

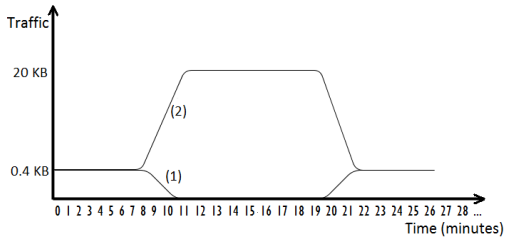

Figure 9. Redundant Traffic Scheme.

Figure 9 shows the precise problem we solved. We have 30 minutes of application running time, from which 10 minutes the user waits. (1) represents the improved traffic evolution and (2) the old one, before the improvement. The traffic consumed drops to zero for the period the user waits.

The same algorithm is used for sending location updates only when the user is waiting, diminishing the traffic almost to zero (except the periods when the user is not moving, which calls for server computation).

Another performance decision was to stop requesting location updates and sending them to the server for the night period. Public transportation system schedules stop at $23:00PM$, so there is no use interrogating the server between $23:00PM$ and $6:00AM$.

VII. Conclusion

Given the current pace of technological evolution, the need to integrate the technology with day by day life is high. Intelligence is at every turn, problems can be solved and there are lots of solutions which involve already existing technology.

Urban conglomerates suffer from pollution, congestion and overcrowding. There is a fast developing trend, called "intelligent transportation", which tries to solve some of nowadays cities' problems. We try to bring our contribution to this field through this project.

The service presented enhances the public transportation systems of modern cities by adding a new level of intelligence. We are using the smartphone, one of the most prevalent gadgets in the world, as an intelligent adviser, ready to provide context-aware relevant information about public transportation. Using the sensing capabilities of the smartphones, our system interprets data, and discovers situations, giving intelligent solutions to problems, answering questions before being asked.

Currently we developed a fully working application fulfilling really useful tasks, enhancing and promoting the use of public transportation. Future work will include studies about performance enhancement, diminishing battery consumption, and, one of the most important things, we aim for a fully integration with the public transportation network, which would be an excellent cornerstone for developing a complete intelligent transportation experience.

Acknowledgment

The research presented in this paper is supported by CyberWater grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012.

References

[1] CakePHP, official website. http://www.cakephp.org/, 2014. [Accessed February 16th, 2014].

[2] Google Now, official website. http://www.google.com/landing/now/, 2014. [Accessed February 16th, 2014].

[3] RATB, official website. http://www.ratb.ro/, 2014. [Accessed February 16th, 2014].

[4] Transport Urban, official website. http://www.transporturban.ro/, 2014. [Accessed February 16th, 2014].

[5] Andrius Aucinas, Narseo Vallina-Rodriguez, Yan Grunenberger, Vijay Erramilli, Konstantina Papagiannaki, Jon Crowcroft, and J Wetherall. Staying online while mobile: The hidden costs. *ACM CoNEXT*, 13, 2013.

[6] Paolo Bellavista, Antonio Corradi, Rebecca Montanari, and Cesare Stefanelli. Context-aware middleware for resource management in the wireless internet. *Software Engineering, IEEE Transactions on*, 29(12):1086–1099, 2003.

[7] Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *Software Engineering, IEEE Transactions on*, 29(10):929–945, 2003.

[8] Anind K Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.