Contextion: A Framework for Developing Context-Aware Mobile Applications

Elizabeth Williams, Jeff Gray
Department of Computer Science, University of Alabama
eawilliams2@crimson.ua.edu, gray@cs.ua.edu

Abstract

Context-aware mobile interfaces that are dynamic and adapt to each user pose a challenge to developers because the interface must continually adapt to accommodate changes in the user's activity and environment. Current methods of development do not allow for efficient creation of contextual applications. In addition, although data from sensors on a mobile device provides a rough estimation of a user's environment, the data needs to be combined in an intelligent way in order to determine a user's intention. In this paper we present the design of a framework called Contextion for easily creating context-aware mobile applications. The framework is built as a layered architecture in order for portions of application components to be adapted based on current contextual information. Contextion also allows for rapid addition of new sensor technologies on a mobile device to the Contextion framework. Using a specification language, end-users can define in what situations various pieces of contextual data should be used and how the data affects the mobile application. In addition, the design of Contextion allows for the definition of operations based on contexts that may not be known at the time of development.

Categories and Subject Descriptors H.1.2 [*User/Machine Systems*]: Human factors; H.5.2 [*User Interfaces*]: User-centered design

Keywords context-aware computing, mobile development

1. Introduction

Context-aware applications take advantage of contextual information to adapt their features to the user's surroundings. Context is defined in several ways. Schilit et al. [6] describe *context* in three aspects: "the location of use, the collection of nearby people and objects, as well as the changes to those objects over time." Dey and Abowd [3] provide a broader definition:

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

The context of a user can include any information that characterizes the situation of the user, including, but not limited to, location and time, as well as a user's emotions, social setting, and surrounding noise level. Dey and Abowd [3] also provide a definition of a *context-aware system*:

> A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

The key challenge in developing context-aware mobile applications is that context constantly changes. Thus, information regarding context must be obtained frequently by a device's internal and external sensors to keep an application current. Chalmers [2] establishes five uses of contextual information:

1. Contextual sensing - where the context is sensed and information describing the current context (e.g. location, temperature) can be presented to the user.
2. To associate context with data, known as contextual augmentation (e.g. records of objects surveyed can be associated with location, meeting notes can be associated with people in the meeting and the place the meeting was held).
3. To enable contextual resource discovery (e.g., to cause printing to be on the nearest printer).
4. Context triggered actions to trigger actions such as loading map data for an area to be entered, or exchange business cards.
5. Contextual mediation - using context to modify a service. For instance to describe limits and preferences over a

MobileDeLi '14, October 21, 2014, Portland, OR, USA..

http://dx.doi.org/10.1145/2688412.2688416

large range of offered data, in order to display the most appropriate parts. The request for the data being mediated need not arise from the context.

To these uses, we have been creating a framework, called Contextion, that will allow developers to produce context-aware applications without the accidental complexity of sensors that collect environmental data. Through our framework, we introduce the concept of intention. Intention is vital to providing a user with relevant information. Yet different users can have different intentions even with the same contextual data. For example, two people might be traveling to the movie theater. Both are driving in a car to the same location at the exact same time. It is probable that one person is going to the movie theater to see a movie, for entertainment purposes. However, the other person might be driving to the theater because that is where he or she works. In this situation, it is clear that both people have contrasting intentions, although contextual data sensed by their mobile devices might be the same. We propose that mining a user's history might give clues to their intention, allowing better adaptivity and contextual awareness. The contribution of our paper is to propose a system that will allow developers to discover a user's intention to create a more accurate context-aware application.

In the next section we discuss the current state of research in context-aware computing. Following, we introduce our framework for creating context-aware mobile applications. We then present an example application that can be developed easily and quickly using our framework.

2. Discussion of Current State of Research

Context-aware computing is an essential but newly emerging area of research. Current state of the art includes various frameworks and technologies that allow developers to utilize data collected from sensors that form a context for a user.

Dey, Abowd, and Salber [4] created The Context Toolkit,

> a framework that supports the acquisition, representation, delivery and reaction to context information that can be automatically sensed and used as implicit input to affect application behavior.

The toolkit was one of the first context-aware frameworks introduced.

Since then, context modeling has been a topic with much potential. According to Bettini et al. [1], there are three approaches to context modeling: object-role based, spatial models, and ontology-based. Out of those, ontology-based models are considered to be the most powerful at expressing complex contextual data and relationships. The OWL-DL language [5] is a popular choice for defining models of contextual information.

However, there are drawbacks to the current state of research:

1. Although some current research does aggregate contextual data, generally a user's motives are not discovered. For example, the context model is different if a user is going to the movies for entertainment, or if the user is heading to the movie theater because he or she works there. We propose that this collected data can be combined more intelligently to discover a user's intention. An intention might be going to a restaurant with friends for fun or cooking breakfast at home in the morning. Knowing a user's intention can allow developers to provide dynamic information that more closely aligns with the user's needs.
2. Current research also does not mine contextual data for characteristics of the user. Contextual inference based also on characteristic information previously mined can improve the context model.
3. Most context modeling algorithm require that there be an initial set of user activities to match against. This means that the user must perform activities at least once before a system can infer it in the future. This is a major drawback because activities that only occur once, such as those performed on vacation, will never have any contextual reasoning by the system.
4. End-users are not generally able to use the existing frameworks that are pervasive in context-aware research. However, end-users can define the most intelligent context models.
5. In the current state of research, developers cannot define operations on contexts that are unknown. For example, if a user goes to a new location that he or she has never been to, the system may not be able to provide any guess as to the user's intention in the new context model. Clustering new context models with old can produce better inferences about the user's motive.

3. Conceptual Framework Design

In this section we discuss our conceptual design for a framework that provides developers and end-users with an accessible and efficient method for creating context-aware mobile applications. The framework is currently designed as an Android library, although it could easily be modified for any platform. We chose Android because of the robustness of the APIs in obtaining contextual information. We first define a sensor used for collecting contextual data:

> A sensor is any piece of hardware or software technology used in the collection of data that forms a context for a user.

Examples of sensors are GPS, accelerometers, and clocks. The user's calendar of upcoming events, apps the user accesses frequently, or social networks can also be considered sensors as they can contribute important data to a user's con-

text. The system can work with as many or as few sensors as the developer requires.

We next define several requirements for the Contextion framework. These requirements address the problems found in the current state of context-aware research:

1. The Contextion framework should allow developers to plug in strategies for combining contextual data intelligently. For example, if a user is at home in the morning, he or she might have a different intention (e.g., perhaps of eating breakfast or getting ready for work), and thus, a different context, than if the user is at home in the evening, when they might be watching TV. Although the location is the same, the combination with the time data created a totally different context.
2. Contextion should allow the building of a user profile, which will hold information and characteristics about the user. For example, a user profile might hold the locations of the user's home and work or places the user frequents. This information can be used for more intelligent inference of contextual models.
3. Users of Contextion should be able to define operations based on the type of context if the specific context is not known. For example, a developer might want an operation to occur when a user is on vacation. It is impossible to enumerate every possible location a user might visit on vacation, so the framework should be able to distinguish which context models indicate that the user is on vacation.
4. Using the Contextion framework, mobile applications should be able to be created without needing to reimplement the collection of contextual data. A specification language will allow developers, and even end-users, to effortlessly build applications that utilize contextual information.
5. Contexts that are unknown (e.g., the user or developer has not been able to explicitly specify the context model) should be clustered with existing context models to provide better predictions about the user's intentions.

The framework is able to automatically detect and formulate a user's context, or environment, through sensors, either built-in or external. It also has the ability to plug in new sensors as technologies are developed. Developers can easily and quickly create new applications that take a user's context into account or infuse existing applications with contextual information and data. The architecture is displayed in Figure 1.

The framework consists of a Sensor interface from which classes of sensors can inherit. Sensor classes that conform to the interface can contribute data to the user's overall context. By hiding the implementation of sensors such as location, time, or accelerometer technologies, developers do not need to be concerned with the realization of obtaining data from these sensors. Currently, developers can write new classes that implement this interface in order to add new sensor technologies as they are developed. Any combination of sensors can be used for an application. Specification files allow developers or users to define rules about various sensors. For example, a developer might specify which data collected from a location sensor represents a user's home and which represents work. These sorts of labels will be stored within the framework and can be accessed by the specification language defined by an application described in Listing 1. Developers could also determine how coarse- or fine-grained data collection for a sensor should be. For example, a developer can decide whether time data should be collected every hour or every minute.

A SensorManager collects and handles all sensor objects within an application. Using the SensorManager, developers need never interact directly with any specific sensor objects. The SensorManager combines all data from the various sensors into one context. The SensorManager also allows developers to plug in strategies for combining raw contextual data into more intelligent context models. These strategies are implemented as Filters. Filters can be implemented that allow the inference of a user's intention or motive within a context model. For example, a pattern matching algorithm could be plugged in to discover patterns in a user's history of contextual data. By discovering patterns, we can fill in absent data if a sensor fails in the collection of data for any reason. Also, patterns can reveal a user's routine, which can enable the recognition of types of information the user might need in various context scenarios.

The last main component in the Contextion framework is the ActivityAdapter. In the Android platform, "an activity is a single, focused thing that the user can do." [1] Typically, a developer creates a new activity for each new screen displayed in the app. These various activities inherit from the Activity class, part of the native Android API. In the Contextion framework, to incorporate contextual information into an application, developers need only inherit from an ActivityAdapter rather than Activity. The ActivityAdapter reads in a specification file and creates various layers of contextual information on top of the existing base activity. Each layer contains data from a single sensor. The ActivityAdapter is responsible for managing contextual information from the sensors, determining when a user's context has changed, and updating the current context model.

Using the Contextion framework, a specification file can be written by a developer or end-user that specifies which data layers are visible at different contexts. An example specification file is shown in Listing 1. In this example, a link is created between a context scenario and an operation. The context consists of the user having a change in location and there being no upcoming events on his or her calendar. When the user is in this context scenario, the app will call

[1] http://developer.android.com/reference/android/app/Activity.html

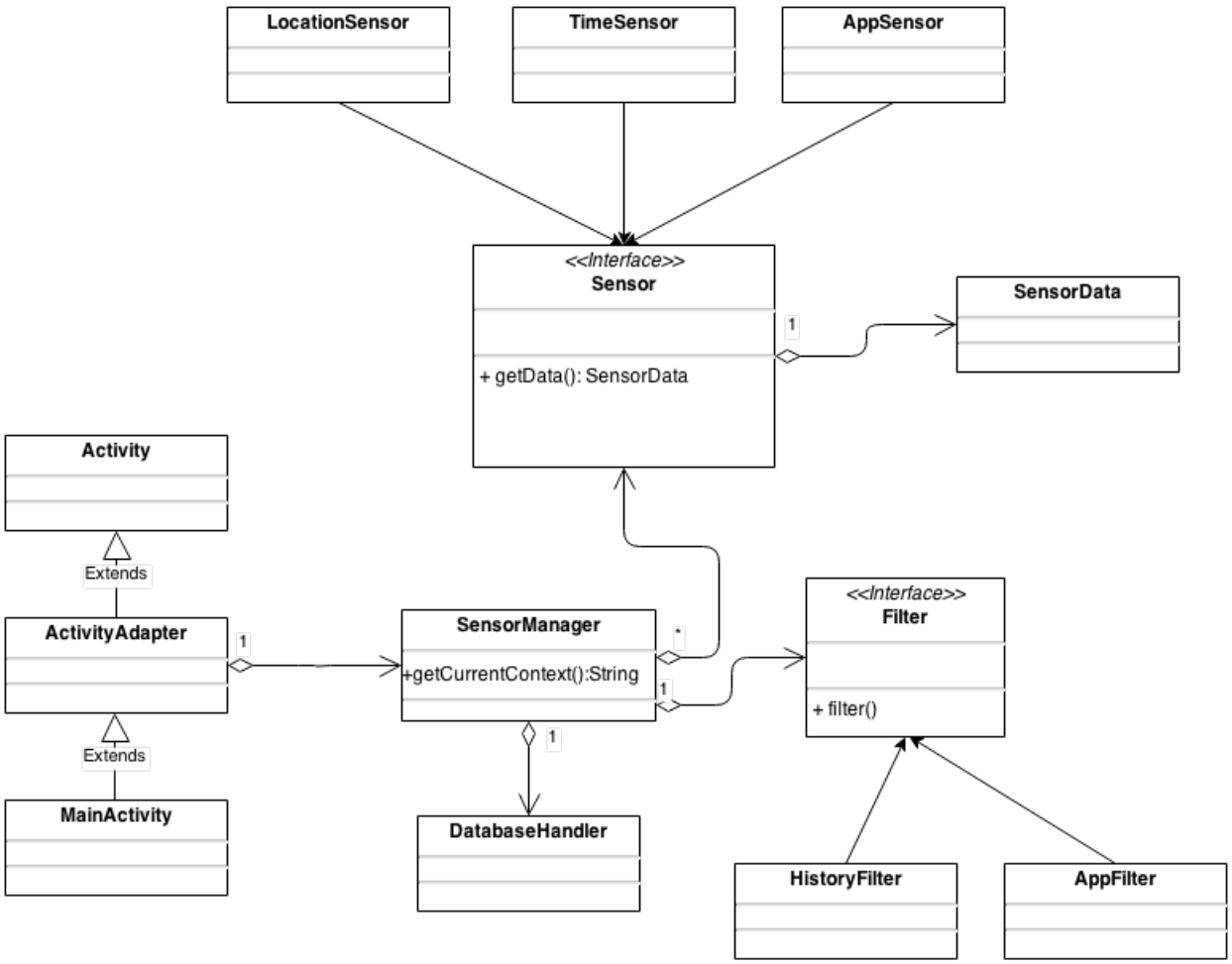

Figure 1. Overview of the Contextion framework.

Listing 1. Specification File

```
<?xml version="1.0" encoding="UTF-8"?>
<file>
    <link>
        <context>
            <location>changeInLocation</location>
            <calendar>null</calendar>
        </context>
        <operation>
            displayNearbyEvents
        </operation>
    </link>
</file>
```

the method *displayNearbyEvents*, which is defined in the application's activity.

Using the Contextion framework, developers do not need to be concerned with the implementation of retrieving data from many different sensors. The developer simply needs to change the inheritance of the application's activities from Activity to ActivityAdapter, write the specification file that defines links between context scenarios and operations, and, in the activity, define the operations that occur in the various contexts.

4. Example Application

In this section we describe a conceptual example application that exhibits the capabilities of the framework outlined in the previous section. This application will filter and display events that might be of interest to the user. Google Now [2] provides a list of events that are occurring soon near the user. Using a location sensor, we can obtain a similar list of events and activities happening nearby the user. We can then use a filter to pattern match between the user's profile and the nearby events.

[2] http://www.google.com/landing/now/

The application consists of only one activity, called MainActivity. To create this application, a developer would need only to inherit from ActivityAdapter from the Contextion library in the MainActivity. This allows the contextual information from a specification file to be used in the application.

The specification file from Listing 1 is used in this application. The file states that when the user changes his or her location and when there are no upcoming events on the user's calendar the *displayNearbyEvents* method should be called. The label of *changeInLocation* would be defined within the sensor specification files described earlier and are referenced here. For example, in the sensor specification file, the *changeInLocation* label might refer to a method that tracks when a user shows significant change in location.

The developer needs to define the *displayNearbyEvents* method in the MainActivity. This method would filter events for those that are determined to be most relevant for the user and displays them when the criteria for the context scenario in the specification file are met.

The ActivityAdapter determines when the user is in the context model defined in the specification file and automatically calls the *displayNearbyEvents* method. The developer need not worry about collecting any contextual data and instead can focus on the changing of the application's user interface with the changing context. The specification file could even be modified by end-users if so desired. This would allow end-users to control their own context scenarios.

5. Future Work and Conclusions

Much research is left in the area of context-aware computing. With the context-aware framework we have presented in this paper we demonstrate how a mobile application can be extended to allow contextual information to be utilized. We introduced the problem of incorporating a user's context, or environment, into a mobile system. We then introduced a framework that allows efficient insertion of sensor data into an application. Finally, we provided an example application that makes use of the Contextion framework.

First, we plan to implement strategies for the SensorManager that can improve inferences about a user's intentions within a context scenario. We intend to look at ways to associate raw sensor data with higher level contexts. In addition, we will look at patterns within a history of these context models as wholes to determine whether better predictions can be made of a user's motives. Finally, we plan to allow users to define operations based on types of contexts rather than simply contexts that are explicity specified. The Contextion framework is already able to handle pluggable strategies for combining and filtering raw data into more abstract contextual models. The framework can also be easily extended to allow context models to hold abstract types upon which developers can define operations.

References

[1] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni. A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.*, 6(2):161–180, apr 2010. ISSN 1574-1192. . URL `http://dx.doi.org/10.1016/j.pmcj.2009.06.002`.

[2] D. Chalmers. Contextual mediation to support ubiquitous computing. Technical report, 2002.

[3] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. In *CHI 2000 Workshop on The What, Who, Where, When, Why and How of Context-Awareness*, 2000.

[4] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.*, 16(2):97–166, Dec. 2001. ISSN 0737-0024.

[5] I. Horrocks, P. F. Patel-Schneider, and F. V. Harmelen. From shiq and rdf to owl: The making of a web ontology language. *Journal of Web Semantics*, 1:2003, 2003.

[6] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, pages 85–90, 1994. .