CAMF – CONTEXT-AWARE MACHINE LEARNING FRAMEWORK FOR ANDROID

Alf Inge Wang[1], Qadeer Khan Ahmad[2]
Dept. of Computer and Information Science, Norwegian University of Science and Technology
Sem Sælandsvei 7-9, N-7491 Trondheim, Norway
[1)] alfw@idi.ntnu.no / [2)] qadeerkh@idi.ntnu.no

ABSTRACT
Context-aware computing is a promising approach for utilizing the characteristics of mobile computing: communication, mobility and portability. By combining machine learning and context-aware computing, we can provide proactive services based on the users' usage patterns of the mobile device combined with the environmental context of the user. Android has become a popular mobile platform, which have addressed context-awareness from day one through hardware and software support for sensor and context management. In this paper, we have evaluated the Android platform support for context-awareness and identified some shortcomings. Further, we have designed a Context-Aware Machine learning Framework (CAMF) for the Android platform that addresses these shortcomings as well as incorporating machine learning. We also demonstrate the usefulness of the framework through the implementation of an application that monitors the applications a user is running on an Android device along with the environmental context the applications are running in. This information can be used to proactively launch Android applications when the context is appropriate. Finally, the paper evaluates the CAMF framework and our context-aware application AppL.

KEY WORDS
Software Design, Mobile and Wireless Computing, Context-aware Computing, and Android platform

1. Introduction

Smart-phones have now become an essential part of daily life. Mobile devices have not only become very powerful computing devices, but they are also equipped with various connectivity features and a wide range of sensors or hardware units that can be utilized as sensors such as accelerometers, GPS, light-sensors, distance sensors, video/photo cameras, microphones, etc. Mobile computing introduces several challenges related to communication, mobility and portability, but it also opens a range of new opportunities such as context-aware computing [1]. Context-awareness can be defined as: *"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"*, where context is *"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"* [2].

A context-aware system can improve the user experience, by only showing or doing the relevant things for a given context. To utilize the context to a large extent, a context-aware application needs to learn how the user use his device and in what context. A learning context-aware system can be achieved through machine learning. Machine learning has many application areas for mobile context-aware applications such as location prediction [3], adaptable user interfaces [4], observing and recognizing human behavior at home [5], recognizing human faces [6], and smart music players that play music according to learned user preferences and activities [7].

The Android platform has good support for developing context-aware systems and is regarded as one of the best choices for mobile application development [8]. Android uses a middleware infrastructure approach to distinct context sensing and usage, supports a range of physical components, and provides software components to offer middleware for the physical components and context preprocessing capabilities. The platform lacks a generalized interface for context management and a discovery component for adaptation. In addition, Android does not include any machine learning capabilities.

In this paper, we have evaluated the Android platform support for context-awareness and identified some shortcomings. Further, we have designed a Context-Aware Machine learning Framework (CAMF) for the Android platform that addresses these shortcomings as well as incorporating machine learning. We also demonstrate the usefulness of the framework through the implementation of the Application Learner (AppL) application that monitors what applications a user is running along with the

environmental context of the applications. This information can be used to proactively launch Android applications when the context is appropriate. Finally, the paper evaluates the CAMF framework and the AppL application.

In this paper we wanted to find answers the following research questions: RQ1) How can context-aware systems benefit from machine learning; RQ2) What context support is provided by the Android platform; and RQ3) How well suited is Android for developing a context-aware system.

The rest of the paper is organized as follows: Section 2 describes related work, Section 3 outlines considerations when designing a context-aware system, Section 4 shows the results of evaluating context-awareness support in Android, Section 5 presents our Context-Aware Machine learning Framework (CAMF), Section 6 describes our experiences from implementing the CAMF framework and a CAMF application named AppL, and finally Section 7 concludes the paper by answering the research questions given above.

2. Related Work

In [9], Dey et al. describe their *Context Toolkit*, which is a framework for rapid prototyping of context-aware applications. The framework is built around the following components: *context widgets* – a software component that provides applications with access to context information from their operating environment, *interpreters* – transforming context information to a higher abstraction level, *aggregators* – collecting multiple pieces of context information that are logically related into a common repository, *services* – executing actions such as changing the state of activators, and *discoverers* – maintaining a registry of existing capabilities in the framework.

In his PhD-thesis, Chen defines an agent architecture for rapid prototyping of context-aware applications [10] based on a context broker that utilizes ontologies for representing and modeling context. It consists of a context knowledge base, a context-reasoning engine, context- acquisition module, and a privacy-management module.

In [11], Baldauf et al. present a survey on various context-aware systems from which they have derived common architectural principles depending on factors such as the sensors are local or remote, the system has many or few users, available resources on the device, and extendibility of the system. Baldauf et al. propose a layered architecture for supporting context-aware systems (see Figure 1) consisting of (from bottom to top) *a Sensor layer* managing physical, virtual and logical sensors, a *Raw data retrieval layer* responsible for providing the layer above with sensor data, a *Preprocessing layer* (optional) improving the quality of sensor data through reasoning and interpretation, a *Storage and management layer* concerned with organizing and providing the gathered data to clients through a public interface and finally a *Application layer* providing a high-level API for context-aware applications.

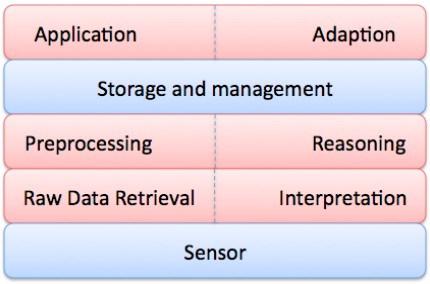

Figure 1. Baldauf's and Miraoui's layered architectures

Miraoui et al. conducted a similar survey with focus on pervasive computing where the focus was on level of context abstraction, communication model, reasoning system, extensibility and reusability [12]. Also Miraoui et al. describe a layered architecture similar to Baldauf (see Figure 1) where the second and third layer from the bottom have been slightly changed: The *Raw data retrieval layer* has been replaced with a *Interpretation layer* that transform crude sensor data into useful context information, and the *Preprocessing layer* has been replaced with a *Reasoning layer* interpret, predict and deduce sensor information from the layer below to produce context information at a higher level. In contrast to Baldauf's architecture, Miraoui's architecture includes an *Adaptation layer*, which is responsible for system adaptation as the environment changes. Note that our mapping is a simplification and that the layers from the two architectures cannot be interchanged.

Persona is a context-aware toolkit for pervasive applications that focuses on end-user personalization and control [13]. The toolkit consists of an interface engine to generate multi-modal user interfaces that support end-user personalization and control, and an API for capturing end-users' interactions. The approach is document centric in such a way that an application's default runtime behavior is overlaid by end-users' preferences.

There are some examples of context-aware frameworks implemented for the Google Android

platform. Shu et al. present the design of a mobile location service for Android to make it easier to manage and query location-aware map services [14]. Tutzschenke and Zukunft have made a context-aware multi-user framework for construction of pervasive games on the Android platform [15]. Hu et al. have made a semantic context management framework for making various types of context information semantically searchable and sharable among context-aware Android applications [16].

CAMF presented in this paper is based on Miraoui and Baldauf's layered architectures and has integrated some ideas from the Context Toolkit. Although the layered approach for context-aware systems has proven to be beneficial, potential problems with this approach must be taken into account such as incomplete rule logic, inconsistent sensing rates, slow sensing, problematic rule logic, overlapping sensor fields, and sensor noise [17]. These problems are usually caused by the nature of layered architecture, the asynchronous propagation of context information and the inaccuracy in the interfaces between layers. There are no simple solutions to avoid this problem, but the applications should be designed with these issues in mind.

The focus of CAMF is different than the existing context-aware framework presented above in that it is a general framework that provides a generalized interface for context management and a discovery component for adaptation. In addition, CAMF is the only context-aware framework for Android that supports machine learning.

3. Design of Context-aware Systems

Context-aware system design is mainly characterized by three factors: a distinction between context sensing and usage, a set of physical components to capture context information, and a set software components to handle and manage context information and to adapt to contextual changes in the environment. In Baldauf et al.'s survey of context-aware systems [11], they identified some common architectural principles. The architecture of context-aware systems depends on factors such as whether sensors are local or remote, whether the system has many or few users, the available resources of the used devices (mobile devices vs. high-end PCs), and the facility of further extension of the system.

Another main architectural driver in context-aware systems is the method of context-data acquisition. Three commonly used approaches are direct sensor access, middleware infrastructure and context server [10]. In the *direct sensor access* approach, the system gathers desired data directly from its sensors without preprocessing or using some other layer. Sensors acquisition is hard-coded in the clients, which gives a tight coupling between the clients and sensors. This approach is often used in devices where sensors are locally built in and not suitable for distributed systems. In the *middleware infrastructure* approach the sensor data acquisition is done through a middleware component. This approach hides low-level sensing details and avoids hard-coding in the clients. It improves reusability of sensors among different clients and simplifies extensibility of clients. The *context server* approach allows clients to remotely access data sources. This approach introduces an access managing remote component, which facilitates concurrent client access of sources. The advantage of this approach is reusability of sensors by remote clients, which also relieves clients from resources intensive operations involved with acquisition. Thus, this approach allows clients to be thin.

CAMF was mainly designed based on the middleware infrastructure approach. This approach was chosen as smartphones are powerful enough to do the extra processing of adding a layer that hides the physical sensors to provide reusability and extensibility. Further, by not choosing the context server approach, the application can work off-line (to save battery and network expenses). The motivation for our framework was based on the following [11]: Strict division of context data acquisition and usage allow context sources to be reusable by multiple clients, the context models and context processing logic is an important criterion for providing intelligent and adaptable context-aware services or applications, resource discovery mechanism is important in dynamic or pervasive environments where available sensors and the context sources change rapidly, and historical context data management combined with learning algorithms can enable proactive services and provide highly adaptable context-aware services.

4. Context-Awareness in Android

Smartphones are ideal for context-aware applications since they are relatively powerful and contain various sensors. Before we decided to go for Android for our context-aware framework, we evaluated other smartphone platforms: iPhone, Symbian, RIM, Windows phone and Linux. The two most promising platforms were iPhone and Android because of the

popularity, high usability, powerful CPUs and available sensors. The Android platform was preferred to iPhone because it uses Java as the main programming language and there were several machine learning frameworks available for Java, it provides access to more core OS functionality, it does not require any certification or developer registration to deploy the software to hardware, and the Android SDK is available on multiple platforms [8].

The context support in Android application framework consists of two main parts: *Raw context data sources* and *Context processing* [18]. The support for *raw context data sources* contains several packages and classes such as for the camera, Bluetooth scanning of nearby devices, sensor manager for controlling interaction with physical sensors on the Android device, geographical location, time, and sound recording. The sensor manager enables Android applications to access a wide range of sensors: accelerometer, light, magnetic field, orientation, pressure, proximity, and temperature.

The context processing support in Android contains functionality for processing raw context data into more useful contextual data and includes face recognition, speech recognition, text-to-speech, location proximity, and a Google Maps API.

The Android application framework provides a very good starting-point for development of context-aware applications, but it lacks a generalized interface for context management and a discovery component for adaption. Further, if proactive context-aware applications are to be developed, machine learning is necessary to recognize previous context patterns.

Separation between context acquisition and usage is very important for context-aware system architectures [11, 12]. Such separation of concern is well supported in the Android application framework through the broker architecture that provides an intent-based communication between components. The Android application framework uses a middleware infrastructure for context acquisition providing interfaces for various sensors in such a way that no data is accessed directly from the hardware. Further, access to remote context servers are supported in Android through various network APIs as well as specific APIs such as for Google Maps.

5. The CAMF Framework

The starting-point for our Context-Aware Machine learning Framework for Android was the layered architectures for context-aware computing defined by Miraoui et al. [12] and Baldauf et al. [11]. The next step was to map the existing context-awareness support in the Android application framework to the layers. The bottom layer, *sensor layer*, is well supported in Android through the raw context acquisition for the various sensors (see previous section). The second layer, *interpretation layer*, is also well supported through the Android middleware components representing the physical sensors and providing reusability of sensors. The third layer, *reasoning layer*, should provide a high-level reasoning and interpretation of context information from the layer below. This is supported in Android by the components for location proximity, face recognizer, speech recognizer and text-to-speech. Android does not provide a complete set of services for this layer, but provides support for some critical context sources (location, picture/video, and audio). The two remaining layers, the *storage and management*, and *adaption* layers, are not supported in Android. Figure 2 shows the architecture of CAMF based on Miraoui and Baldauf's layers (see Figure 1).

The *processing widgets* are based on the context widgets from the Context Toolkit [9]. The goal of the widgets is to hide the complexity of the sensors used by the application by processing contextual data from single or multiple context sources. To further improve the preprocessing layer, our framework includes machine-learning support for widgets. This means that the widgets can build a database of context data, which they can feed to machine learning algorithms to acquire useful classifiers. This approach can e.g. be used to find whether the user is indoors or outdoors by using a trained classifier of context data such as surrounding sounds, environmental temperature, light and location. Similarly, another widget can use machine learning to recognize whether a certain person or object is nearby (like a car), by using sound characteristics of the person or object.

The *generalized interface* provides a context interaction scheme using a mutual interface for both processing widgets and raw context layer components. This interface can be used by applications to request or subscribe to context events. This will simplify the context handling since developers spend less time learning widget and context source interfaces, and context sources and widgets can be easily replaced in the application. The generalized interface also provides a storage of context data atoms (single context data value) [11] to provide support for context history. Historical context data is useful in applications and widgets that incorporate machine learning because of the way classifiers are trained using training examples.

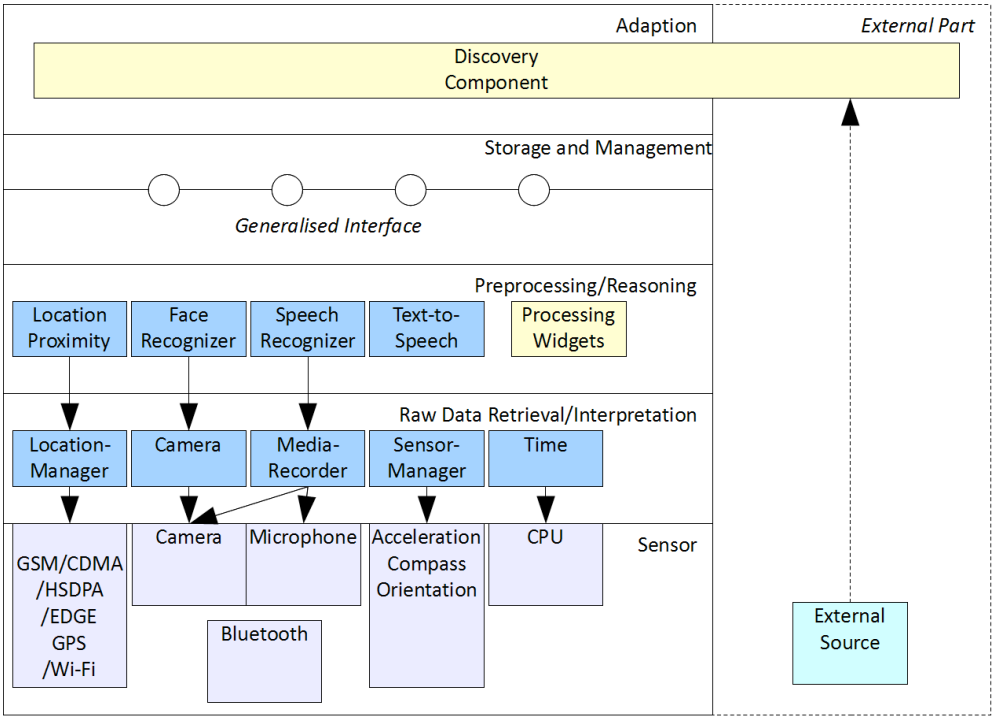

Figure 2. CAMF layered architecture

The *discovery component* is an adaption layer that can discover and discard context sources dynamically, including external previously unseen context sources. All the sources that are available for the application will be published in this layer in real-time. For internal context sources, an adaption controller component in the adaption layer can automatically disable or enable context sources as the accuracy drops or increases. For external context sources, this is much harder – but this can be implemented through the available PAN and WAN technologies provided on the Android device.

6. Implementing and Evaluating CAMF

This section describes some implementation details of the CAMF framework, implementation of a CAMF application and an evaluation of the framework and the application.

6.1 Implementing the CAMF framework

Our current implementation of the CAMF framework contains three main modules: The *context source module* providing a generalized interface, the *database module* providing database support for context information, and the *Weka service module* providing machine learning capabilities to be used with context information. The adaption layer has not been implemented yet.

Figure 3 shows an overview of the classes in the *context source module*. `SourceChangeListener` must be implemented by any component that wishes to receive source updates. The main purpose of `ContextSource` is to require that context sources, both raw sources and processing widgets, implement a `requestSourceUpdate()` method, which `SourceChangeListener` can use to request updates. After a call to `requestSourceUpdate()`, `ContextSource` calls the `onContextSourceUpdate()` when its source data is ready. The `ContextSourceData` is used by context sources to return requested context data. This interface requires context sources to return data with at least the following properties: 1) actual data in a String array, 2) time stamp, and 3) the `ContextSource` object that created this data. These properties follow the suggestion of context data atom attributes of Baldauf et al. [11] by including: Context type (found through the source), context value, time stamp, and source. In order to be saved in a database, `ContextSourceData` and `ContextSource` implement `ContextDBEntry` and `ContextDBComponent` respectively.

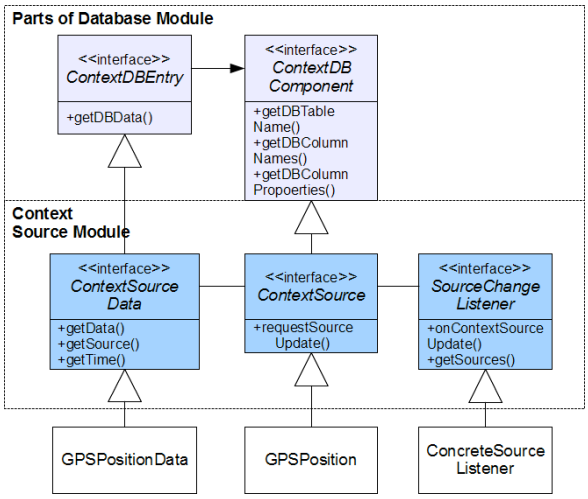

Figure 3. Overview of the context source module

The *database modules* is a SQLite database used to save context data, context source information and their relationship to context listeners, interfaces for objects that can be saved in the database, and a database adapter to fetch and insert data from/to the database.

The *Weka service module* is used to access machine learning capabilities using the Weka open source collection of machine learning algorithms [19]. Weka achieve machine learning independent of algorithms by using datasets – sets of data instances like training data, classifiers – an abstract class all machine learning algorithms derive from, and filters – that transforms datasets my removing/adding attributes, re-sampling the dataset, removing examples etc. The Weka framework had to be ported to the Android platform to be integrated into the CAMF, as it is implemented for the Java platform.

6.2 Implementing a CAMF application

To test the usefulness and the performance of the CAMF framework, we implemented an application we named Application Learner (AppL). The main tasks of AppL is to learn when users start other applications by collecting context data and creating a classifier, and to use the learned classifier to start applications in the right context. Such an application have several useful application areas, such as the mobile device knows when its sound should be turned off (e.g. it detects that the user is in a church or in the movie theater), the mobile device will automatically search for the top 10 movies when the user is approaching the movie theater, or that an alarm should be set of for a given context.

The AppL implementation is focused around five activities: 1) User can choose the application to monitor (learn context from), 2) Cancel monitored application, 3) Create a classifier from the training examples of the application, 4) Context-based application launcher that monitors application context and use a selected classifier (provided by Weka) to predict whether the application should be started or not, and 5) Cancel active application that have been selected to be launched.

Figure 4 shows screenshots from 6 steps of initiating training of a watch ("Klokke") application in AppL: 1) Choose "Train new" to start training an application, 2) Choose application to train, 3) Choose context sources to monitor, 4) The context monitoring of applications has started, 5) The user can get a list of all applications being monitored, and 6) The "Watch" application is being monitored visible as the AppL is getting current location (the ring in the screenshot indicates that the GPS is being accessed).

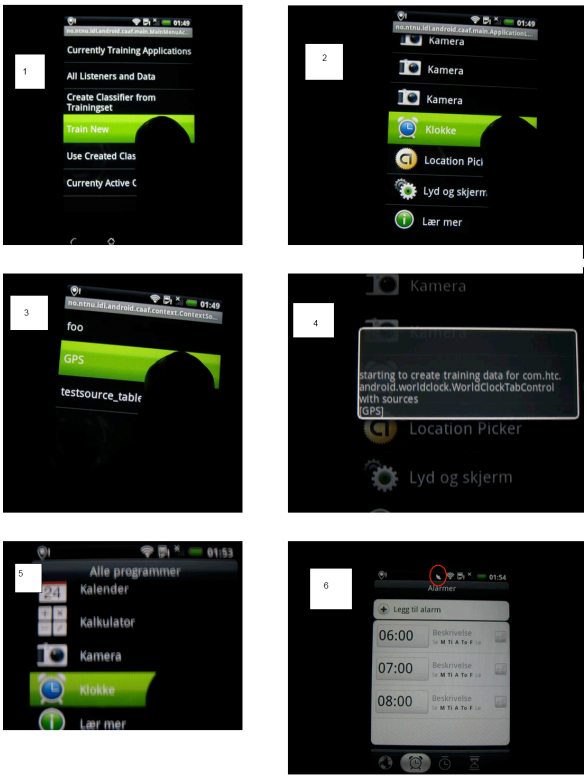

Figure 4. Screenshots from initiating training

6.3 Evaluation of the AppL and CAMF

One of the main tasks of AppL was to discover what applications the user initiates, and a polling service

was used to poll the current active applications to decide whether a given application had been initiated. The main disadvantage of this approach is that the AppL application uses extra resources and thus drains battery by polling the various applications all the time. Unfortunately, we did not find any alternatives to avoid this problem, but we found better ways of polling on Android (by using the Android `AlarmManager`) [20]. Another potential problem with our polling approach is that there is no way to explicitly quit applications in Android and therefore to know when the user quits an application. Android OS closes applications automatically when appropriate. However, from experience we found that Android closes most applications (apart from the Android Browser) when the back button is used to exit an application.

One of our worries when implementing the CAMF framework was the performance of the Weka machine-learning framework on a mobile device. Performance tests from running AppL on a HTC Hero (528 MHz CPU, 288MB ram) showed that learning less than 30 training examples took on average about 1,5 seconds. This performance was better than we anticipated, as the Android device we used was not the most powerful on the marked. As the Android devices become more and more powerful, 1GHz CPUs and beyond, the machine learning performance should not be a big issue. However, for many training examples, we recommend to run the machine learning services on a server. An extension to CAMF would be to include a dynamic load balancing service that decides whether machine learning should be processed locally or remote based on the data set to be processed, available network connectivity and bandwidth, energy level etc [21].

Weka uses supervised learning to train a classifier from a set of training examples. In a training period on AppL, whenever a monitored application is started, it is registered in the database as a positive example, i.e. a training example where the state *application starting* was true. Currently, there is no support for adding negative examples, i.e. examples where the application starting should be false. This is because it is hard to find out when a user does not want to start an application from usage pattern. We think that AppL may benefit more from reinforcement learning [22] than supervised learning. In reinforcement learning, when the machine-learning algorithm makes a prediction, a teacher provides feedback on how good or bad the prediction was. The algorithm then uses this feedback to make better predictions in future. In AppL this can be used to receive feedbacks from the user whenever AppL starts an application, which can ultimately result in better prediction. We initially tried to find a reinforcement-learning framework for Java to integrate with Android, but were unsuccessful.

A form of reinforcement learning may be achievable using supervised learning. Whenever AppL makes a prediction of starting an application, the user could supply a feedback on whether it was correct or incorrect prediction. This feedback, together with the sources' values can be added to the application's dataset as a new training example. Rebuilding the classifier will then, most probably, result in better predictions. By using this approach we are also able to acquire negative examples.

We found the CAMF framework very useful when we implemented AppL. By using the `SourceChangeListener` interface, context listeners can request data from CAMF sources and use the database adapter to read and write to the database. Further, the Weka service in CAMF was an important tool for using machine learning to create proactive behavior in AppL. The current implementation of CAMF demonstrates the main functionality we wanted in our context-aware machine-learning framework. The implementation of the framework is still in an early phase, and there are many areas to expand and improve such as the more processing widgets, implementation of the discovery component (adaption layer) and a load balancing service.

7. Conclusion

Based on our experiences from designing and implementing the CAMF framework and the AppL application for Android, we have found the following answer to our research questions listed in Section 1. R*Q1)* How can context-aware systems benefit from machine learning: Machine learning provides mechanisms to proactively discover new resources and make the system adaptable, and it can be used to develop a reasoning system that infers new context data from raw context data. Smartphones were also proven powerful enough to process machine learning.

RQ2) What context support is provided by the Android platform: The Android platform provides components and interfaces for raw context data and context processing. The support for various sensors is not complete (e.g. RFID is currently not supported and a general object recognizer for video or pictures are not in place), but the mostly used context sources are supported.

RQ3) How well-suited is Android for developing a context-aware system: Android is a very promising

platform for context-aware systems, as Android devices supports a wide variety of physical sensors, the Android application framework uses a middleware infrastructure approach that separates context acquisition and usage and the sensor layer and the raw retrieval layer is well supported. Finally, Android programming API is very close to traditional Java, which means that context-frameworks implemented in Java such as Weka can be ported. We have also identified several support for context-awareness improvements for the Android: platform: 1) context preprocessing widgets together with machine learning provides reusable and advanced context preprocessing, 2) a generalized interface for storage and management to simplify high-level context handling, 3) store historical context data to support machine learning, and 4) provide discovery component for dynamic discovery of external and internal context sources. The CAMF framework was designed to address these improvements.

References

[1] G. H. Forman and J. Zahorjan, "The Challenges of Mobile Computing," *Computer,* vol. 27, pp. 38-47, 1994.

[2] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a Better Understanding of Context and Context-Awareness," in *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing* Karlsruhe, Germany: Springer-Verlag, 1999.

[3] T. Anagnostopoulos, C. Anagnostopoulos, S. Hadjiefthymiades, M. Kyriakakos, and A. Kalousis, "Predicting the location of mobile users: a machine learning approach," in *Proceedings of the 2009 international conference on Pervasive services* London, United Kingdom: ACM, 2009.

[4] A. Shankar, S. J. Louis, S. Dascalu, L. J. Hayes, and R. Houmanfar, "User-context for adaptive user interfaces," in *Proceedings of the 12th international conference on Intelligent user interfaces* Honolulu, Hawaii, USA: ACM, 2007.

[5] D. H. Wilson, A. C. Long, and C. Atkeson, "A context-aware recognition survey for data collection using ubiquitous sensors in the home," in *CHI '05 extended abstracts on Human factors in computing systems* Portland, OR, USA: ACM, 2005.

[6] M. Davis, M. Smith, J. Canny, N. Good, S. King, and R. Janakiraman, "Towards context-aware face recognition," in *Proceedings of the 13th annual ACM international conference on Multimedia* Hilton, Singapore: ACM, 2005.

[7] S. Dornbush, A. Joshi, Z. Segall, and T. Oates, "A Human Activity Aware Learning Mobile Music Player," in *Proceeding of the 2007 conference on Advances in Ambient Intelligence*: IOS Press, 2007.

[8] S. P. Hall and E. Anderson, "Operating systems for mobile computing," *J. Comput. Small Coll.,* vol. 25, pp. 64-71, 2009.

[9] 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.,* vol. 16, pp. 97-166, 2001.

[10] H. Chen, "An Intelligent Broker Architecture for Pervasive Context-Aware Systems." vol. PhD Baltimore: University of Maryland, 2004.

[11] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *Int. J. Ad Hoc Ubiquitous Comput.,* vol. 2, pp. 263-277, 2007.

[12] M. Miraoui, C. Tadj, and C. b. Amar, "Architectural survey of context-aware systems in pervasive computing environment," *Ubiquitous Computing and Communication Journal,* vol. 3, 2008.

[13] F. Kawsar, K. Fujinami, T. Nakajima, J. H. Park, and S.-S. Yeo, "A portable toolkit for supporting end-user personalization and control in context-aware applications," *Multimedia Tools Appl.,* vol. 47, pp. 409-432.

[14] X. Shu, Z. Du, and R. Chen, "Research on mobile location service design based on android," in *Proceedings of the 5th International Conference on Wireless communications, networking and mobile computing* Beijing, China: IEEE Press, 2009.

[15] J.-P. Tutzschke and O. Zukunft, "FRAP: a framework for pervasive games," in *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems* Pittsburgh, PA, USA: ACM, 2009.

[16] D. H. Hu, F. Dong, and C.-L. Wang, "A Semantic Context Management Framework on Mobile Device," in *Proceedings of the 2009 International Conference on Embedded Software and Systems*: IEEE Computer Society, 2009.

[17] M. Sama, D. S. Rosenblum, Z. Wang, and S. Elbaum, "Multi-layer faults in the architectures of mobile, context-aware adaptive applications," *J. Syst. Softw.,* vol. 83, pp. 906-914.

[18] Google, "Android Developers, http://developer.android.com/." vol. 2010.

[19] R. R. Bouckaert, F. Eibe, M. Hall, R. Kirkby, P. Reutemann, A. Seewald, and D. Scuse, "Weka---Machine Learning Software in Java, URL: http://sourceforge.net/projects/weka/files/." vol. 2010, 2010.

[20] M. Murphy, "Diamonds are forever, services are not, Web: http://www.androidguys.com/2009/09/09/diamonds-are-forever-services-are-not/." vol. 2009, 2009.

[21] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing Performance, Energy, and Quality in Pervasive Computing," in *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*: IEEE Computer Society, 2002.

[22] T. Mitchell, *Machine Learning*. San Francisco: Ignatius Press, 1997.