# Architectural Solutions for Context-Aware Applications: KoDA Prototype

Dennis Lupiana
*Faculty of Computing, Information Systems and Mathematics*
*Institute of Finance Management, Tanzania*

## Abstract

*Central to any context-aware application is a context-aware architecture. A context-aware architecture monitors and analyzes its environment to enable context-aware applications and subsequently computing devices to effortlessly and appropriately respond to users' computing needs. This paper presents a Knowledge-driven Distributed Architecture (KoDA), describes implementation of its prototype and illustrates how it can be applied and used in a real-world environment. KoDA is unique as is developed based on a knowledge intensive model, which provides a comprehensive model of the real-world in which the users and computing devices interact. Thus apart from monitoring ability, KoDA can intelligently use relevant information that is accessible in a room to recognize the users' ongoing context. KoDA collectively takes into account information about who, what, when and where to recognize ongoing context.*

## 1. Introduction

In today's dynamic computing environments, context-awareness computing plays a significant role on making the use of computing devices intuitive and less intrusive. Context-awareness computing is a computing paradigm that focuses on inventing applications that are aware, responsive and adaptive to their environments. These applications are called context-aware applications. Center to context-awareness are context-aware architectures. These architectures provide a complete package for monitoring a physical environment, analyze it and share inferred knowledge with context-aware applications. This in turn makes computing devices to effortlessly and appropriately respond to users' computing needs.

By context, I mean any social setting, such as a meeting, where participants aim to achieve a goal. This definition is in line with Schilit *et al.* [11] who argued that context is a much more powerful concept and pleaded with researchers to focus on a broader view of context. This definition emphasizes meaningful interactions between relevant entities required to describe the real-world that is of interest to users and their devices [12]. In this regard, an input to a context-aware architecture is referred to as a *context parameter*. Thus, a context-aware application is a computer application that responds accordingly to social gathering.

It is estimated that each person will have more than six devices by the year 2020 [1]. This is so because mobile devices are increasingly becoming more powerful and sophisticated, thus enabling people to accomplish more while are on the move. As more computing devices emerge, however, interacting and using them becomes difficult and more time consuming. Users and their devices enter and leave different environments where different settings and computing needs may be required. To effectively use devices in such environments means to constantly be aware of their whereabouts, functionalities, and desirable working conditions. Thus, realizing context-awareness is very crucial.

Apart from context-aware architectures, other architectural solutions have been offered to realize context-awareness. Researchers have also responded by developing context-aware middleware and frameworks. Context-aware middleware [8-10] focus on facilitating sharing of context information between context-aware applications hosted in heterogeneous mobile devices while context-aware frameworks [5-7] seek to provide library of software modules to simplify development of context-aware applications. Initial efforts attempted to develop context-aware applications but often focused on automating a specific task and thus only few aspects of the real-world that were relevant to the solution were considered [2-4].

This paper focuses on context-aware architectures and therefore discussion of initial context-aware applications, context-aware frameworks and context-aware middleware is not covered. In particular, this paper presents a Knowledge-driven Distributed Architecture (KoDA), describes implementation of its prototype and illustrates how it can be applied and used in a real-world environment. The rest of the paper is organized as follows. The description of KoDA, its components and how they interrelate is provided in section 2. To illustrate how KoDA can be implemented and used, a prototype is developed. The description of the prototype is provided in section 3. Section 4 illustrates how KoDA can be applied and used in a real-world environment while section 5 provides a summary and a conclusion.

## 2. Conceptual design of KoDA

As proposed by Coutaz et al. [13] and like other context-aware architectures, KoDA is designed as a three-layer architecture. As shown in figure 1, KoDA consists of *perception layer*, *inference layer* and *application layer*. The perception layer monitors an environment while the inference layer uses available information from the environment to recognize ongoing context. The application layer shares the inferred knowledge with context-aware applications for them to respond accordingly. The design of KoDA is largely influenced by a Knowledge-intensive Context Model (K*i*CM) [12]. The following is a brief description of each of the layers:

### 2.1. Perception layer

This layer establishes connections (with both physical and logical sensors), acquires and interprets data from sensors. To implement these capabilities, this layer is designed as an interplay of four components; *sensors*, *sensor platform*, *context interpretor* and *concept base*. The rest of this section describes each of these components.

**2.1.1. Sensors.** To monitor its environment, KoDA is designed to use various sensors. These sensors can be *physical* or *logical* sensors. Physical sensors are hardware sensors such as Bluetooth and RFID[1] while logical sensors are other sources of data such as applications that monitors network activities or CPU usage. Since K*i*CM requires each of its entities to be used, KoDA is designed to monitor all the entities specified in K*i*CM. As is described in the sensor platform, KoDA is designed to easily add new sensing technologies as they emerge. For a discussion on relevant sensing technologies in Context-Awareness refer to Schmidt et al. [14].

**2.1.2. Sensor platform.** To accommodate sensing technologies as they emerge, KoDA is designed with the sensor platform. The platform is a middleware which separates sensors from data interpretation process. The platform is responsible for discovering and establishing connections with and acquiring data from sensors. The platform is designed with an array of modules required by different sensors in order to be used by KoDA. Therefore, the platform is a bridge between sensors and the *context interpretor*. This abstraction enables sensors to be added or removed without affecting other components and hence makes KoDA scalable and flexible.

**2.1.3. Context interpretor.** For a context-aware architecture to recognize ongoing context, it should be aware of different aspects of its environment. In KoDA, this is achieved by monitoring changes within the environment. Sensors, through the sensor platform, acquire data about different aspects of the environment. The interpretor interprets this data to context parameters, aggregates and hands over the context parameters to the *inference engine*, which is a component of the inference layer. To interpret the data, the interpretor utilizes the knowledge stored in the *concept base*.

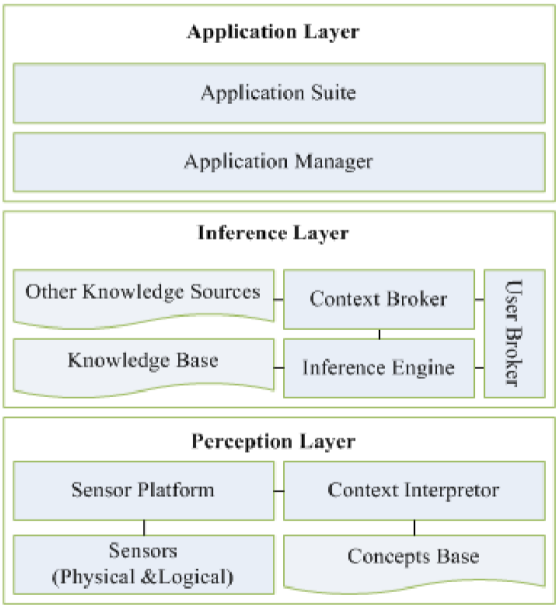


Figure 1. Conceptual design of KoDA

**2.1.4. Concept base.** In practice, a context-aware architecture is implemented in a real-world environment where sensors acquire data specific to that environment. To interpret this data, the architecture should have a prior knowledge about entities in the environment. This knowledge is represented using a context model designed for that architecture. In KoDA, such knowledge is stored in the concept base and developers use K*i*CM to represent it [12]. Among others, the concept base stores knowledge about the mapping between the users' true identities and the devices used to identify the users.

### 2.2. Inference layer

To exploit information, or evidence, collected from an environment, a context-aware architecture should be able to reason about this information. In KoDA, this is achieved by the inference layer. This layer utilizes the information from the perception layer, knowledge about typical contexts within an environment, and its reasoning capabilities. To realize these capabilities, the layer is designed as an interplay of two components; *knowledge base* and an *inference engine*. The rest of this section describes each of these components.

[1] RFID is an acronym for Radio Frequency Identification

**2.2.1. Knowledge base.** For a context-aware architecture to exploit evidence collected from an environment, it should possess knowledge about that environment. In KoDA, this knowledge is represented based on K*i*CM and stored in the knowledge base as inference rules, cases or a Bayesian network. KoDA uses this knowledge and the evidence to infer ongoing context. Thus, the comprehensiveness of a context model is crucial for the richness of knowledge and subsequently for the ability of a context-aware architecture to perceive its environment to enable context-aware applications to adapt to social dynamics.

**2.2.2. Inference engine.** The inference engine is the "*brain*" of KoDA. The inference engine is responsible for inferring ongoing contexts. Based on evidence collected from an environment, the inference engine assigns truth values to the knowledge stored in the knowledge base based on specified constraints. The engine assigns *true* truth values when the constraints are met and *false* truth values when the constraints are not met. The *true* truth values mean that a context that matches the collected evidence is found. This means KoDA has recognized ongoing social context of the users.

In logical-based inference techniques, inference engine terminates the inference cycle if one or more constraints do not match the collected evidence. In probabilistic inference techniques, inference engine assigns low probability to a recognized context. In the real-world, this means that architectures will not provide any computing services or will provide default services. Although this can be the best alternative, it can annoy users and hence make them reluctant to use context-aware applications. To remedy this problem, KoDA is designed to seek knowledge from other sources such as Webpages, electronic calendars and social media. To accomplish this, the layer is designed with two additional components; *context broker* and *user broker*.

- *Context broker.* The context broker is responsible for establishing connections with alternative sources, acquiring knowledge and representing it to the *application manager* of the application layer. To accomplish this, the broker refers to the knowledge about individual entities from the concept base. The broker, for instance, may acquire calendar entries from the users' personal Website or Smartphone. In case there is no alternative knowledge, the broker notifies the user broker.
- *User broker.* The user broker is responsible for managing communications between KoDA and users. KoDA is designed to request users to specify their ongoing context in case there are no matching prepositions in the knowledge base and no knowledge from alternative sources. The user broker listens from the inference engine and context broker for any request of feedback from users. The user broker sends the feedback to the application manager. Depending on the feedback, the application manager provides users with default applications or applications that are appropriate to users' ongoing context.

## 2.3. Application layer

KoDA uses available information within an environment to recognize ongoing context and subsequently enable context-aware applications to respond appropriately. Like the majority of the existing context-aware architectures, KoDA is designed with an application layer. This layer is responsible for invoking applications based on the users ongoing context. To achieve this, the layer is designed as an interplay of two components; *application manager* and *application suite*. The rest of this section describes these components.

**2.3.1. Application manager.** This is responsible for executing appropriate applications depending on a recognized context, as inferred by KoDA or as specified by users. In case no context is recognized and there is no knowledge from alternative sources and feedback from users, the manager will invoke default applications as designated by developers. If the recognized context is a formal meeting, for instance, the manager searches for appropriate applications and executes them. Typical inputs the manager accepts include a recognized context, the users with details of their computing devices and the venue.

**2.3.2. Application suite.** This is where the available applications exist. Applications can exist independently or as a suite of applications for a specific function. Applications can be executed from a user's devices and/or a specialized application server. An application from a Smartphone, for instance, can be executed to remotely change the alerting mode from ringing to silence. Likewise, an application from an application server can be executed to automatically power ON or OFF a projector within a particular venue.

The separation of the application manager and the application suite enables either part to be modified without affecting the other. A new application, for instance, can be added in KoDA without affecting how the application manager operates. Likewise, the application manager can be modified without affecting the operations of the applications. This design philosophy enables KoDA to be easily modified and hence more applications can be added as needs arise. This, as a result, makes KoDA flexible and scalable.

KoDA is a distributed architecture and hence client-server architecture is adopted. In KoDA, a client is referred to as a *proxy computer*. A proxy computer is a computer located in a venue that connects all none IP hardware sensors in the venue. The purpose of proxy computers in KoDA is to facilitate communications between none IP sensors and the server. In KoDA a server is a powerful computer located in any room within a workplace. Ideally, each room in a workplace will have one proxy computer. Both the server and proxy computers are connected to and communicate through a Local Area Network (LAN).

## 3. KoDA prototype

A prototype was developed and used in a real-world environment to illustrate the capabilities of KoDA. The prototype is developed to illustrate how KoDA can exploit available information within a venue and a prior knowledge about contexts to autonomously recognize ongoing context. It should be noted that the prototype is not developed to illustrate completeness of KoDA and therefore few sensors were used to monitor few aspects of the entities specified by K*i*CM.

### 3.1. Knowledge representation

Knowledge of contexts and relevant context parameters can be acquired using different knowledge acquisition techniques. In this work, observation technique has been used because the researcher was part of a research group that some of its members were involved in the experiments.

To represent knowledge of context parameters, eXtensible Markup Language (XML) has been used. Alternatively, Ontology can be used but is more useful when software modules from different vendors, and which have different semantics, interact. For software modules developed with similar semantics, as in this work, XML is suitable and hence preferred in this research. The XML document is stored in the concept base of the perception layer of KoDA.

For KoDA to reason about and recognize ongoing contexts, knowledge of contexts need to be represented or encoded in the KoDA and in particular in the *knowledge base* as inference rules. To achieve this, both rule-based Knowledge Representation Language (KRL) and Bayesian network have been used. Due to limited space, however, only knowledge representation using rule-based KRL is described in this paper. In this language, knowledge about contexts is represented as an IF THEN rule. As shown in figure 2, the context parameters are represented as patterns of conditions at the left-hand side of the rule while their relationships are maintained by logical operators. The right-hand side of the rule specifies actions to be invoked when the conditions are satisfied.

### 3.2. Environment monitoring

To monitor a physical environment, one physical and four logical sensors were used. For physical sensor, a pair of an RFID reader (figure 3) and an antenna (figure 4) was used. The reader was given a unique ID that was associated to a venue. Users were given RFID cards and each was associated with a unique number of an RFID card. Therefore, if Bob is associated with card ‘x’ and the signal of the card is picked by the reader, then the prototype concludes that Bob has entered the room.

Figure 3. High frequency mid-range RFID reader

Figure 4. High frequency loop antenna

For logical sensors, I used the built-in clock of a computer and developed three applications. I used the clock to read system’s time. I developed Keyboard Activity Monitor (KAM) and Mouse Activity Monitor (MAM) to monitor users’ keyboard and mouse activities respectively. I developed another application for monitoring computing devices to determine whether they are ON or OFF.

### 3.3. Data interpretation

The prototype utilizes the *context interpretor* to interpret data captured by the sensors to make facts about the environment, which are required by the prototype to recognize ongoing context. To achieve this, first the data is mapped to context parameters of the relevant entities. Unique numbers of RFID cards, for instance, were mapped to the real names of the people who participated to the experiment. As described in section 3.1.4, the knowledge for mapping is stored in the *concept base*. Thus, the context interpretor interprets the data to context parameters, aggregates them and hands them over to the *inference engine*.

```
rule "formal_meeting_1.0"
    when
        $roomResidents: List()
        $userStatus: List()
        $time: Time(timeFor != "after work")
        Room($venue: roomname, roomcategory == "Meeting Room")
        $device: Device(type == "Projector", room == $venue, status == "ON")
        $userIn: User()
        ArrayList(size >= 1) from collect (User(userrole == "Advisor") from $roomResidents)
        ArrayList(size >= 1) from collect (User(userrole == "Guest") from $roomResidents)
        ArrayList(size >= 1) from collect (User(userrole == "Supervisor") from $roomResidents)
        ArrayList(size == 0) from collect (User(userrole == "Student") from $roomResidents)
        forall (User() from $roomResidents
                UserStatus(pname[0].processname == "keyboard", pname[0].status in
                (null, "false", "true"), pname[1].processname == "mouse", pname[1].status in
                (null, "false", "true")) from $userStatus)
    then
        AppManager applicationManager = new AppManager();
        applicationManager.turnCompOFF("Formal Meeting", $userIn.pcMAC, $userIn.pcIP);
        System.out.println("Time: " + $time.time + ", Venue: " + $venue + ", Social Context: "
         + "Formal Meeting");
end
```

Figure 2: Rule representing a 'formal meeting' context

The prototype utilizes XML document to represent the knowledge in the *concept base*. To retrieve relevant knowledge from the XML document, the Document Object Model interface has been utilized to implement the context interpretor. Since the knowledge is about a specific entity, Java objects have been created for each entity. Java arrays have also been implemented for each of these objects in order to temporarily store knowledge of different instances of the entities. When the data from a sensor is received, the interpretor retrieves relevant knowledge and creates relevant objects. These objects are then inserted into relevant arrays ready to be inserted into the inference engine.

### 3.4. Knowledge reasoning

The prototype utilizes *Rete* algorithm [15] to implement the *inference engine*. The algorithm outlines procedures required to match patterns with the observed facts. These procedures are *match*, *select* and *act*. In *match*, the algorithm compares the patterns of each of the existing rules with a set of facts. The possible outcome of this procedure is (i) no match, (ii) one match or (iii) more than one match. *Select* analyzes the output in order to halt the engine (if there is no match) or to choose a rule that should be executed. *Act* executes instructions specified in the consequence part of a rule.

Despite being an efficient pattern matching algorithm, *Rete* algorithm is preferred because it has been successfully used in context-awareness before. There are a number of rule engines that implement the *Rete* algorithm such as Zilonis[2] and Jess[3]. This prototype, however, preferred Drools rule engine because it offers an explanation facility, which is very useful for performance evaluation of KoDA.

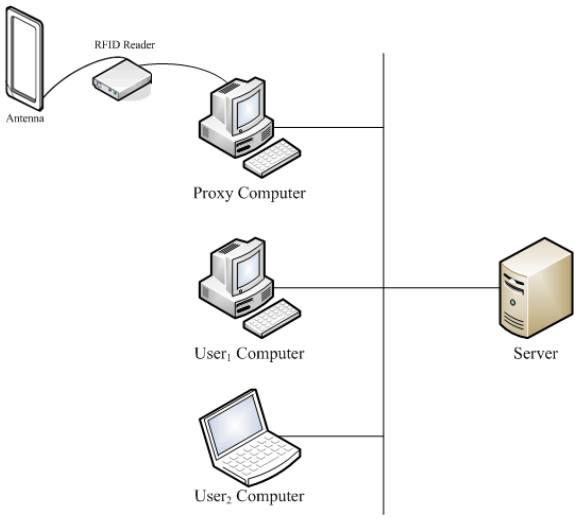


Figure 5. Conceptual representation of the prototype

Figure 5 provides a conceptual representation of the prototype. As shown in the figure, the antenna is connected to the reader and the reader is connected to the proxy computer. The proxy computer is then connected to LAN where it communicates with the server. The figure also shows users' computers are connected to LAN. Each of the user's computers is installed with KAM and MAM and communicates with the server only when requested by the server. For more on this please refer to figure 6.

To enable the proxy computer and the server to communicate over the LAN, *socket programming* was used. I also used *event-driven programming* to develop a program that enables the proxy computer to listen from the reader continuously and send data

[2] http://www.zilonis.org/
[3] http://herzberg.ca.sandia.gov

acquired to the server. The data is acquired when an RFID card is pointed over the antenna. In practice, the data is acquired when a user enters the room. Before this data is sent to the server, the program appends a numeric value that identifies the reader and subsequently the room. If there is no data from the reader (i.e. nobody enters the room) in the last five minutes, the program automatically generates a notification message and send it to the server. As shown in figure 6, this message is used by the server to request feedback of the users' keyboard and mouse activities from KAM and MAM respectively.

The server's program is also developed to continuously listen from and respond to data received from proxy computers. As illustrated in figure 6, the server initializes required software modules and interprets the data received to determine the venue. If the input is a notification message, the server checks if there is anyone in the venue. If there is someone, the server checks the status of their keyboard and mouse activities and determines their ongoing context. If the input is data from the reader, the server loads the corresponding information. This information includes name of the user who entered the venue, her office, role, social relations and other devices the user may own and their IP addresses. The server also checks the status of the keyboard and mouse activities of the users who are in the room. The server uses this information to determine the ongoing context of the users.

As shown figure 6, the server requests for feedback of the users' keyboard and mouse activities from KAM and MAM respectively when (i) a user enters in the room or (ii) after every five minutes from the time the first user entered the room. To address a similar problem, Sensay [16] used a ten-minute duration while Harter *et al*. [17] used a five-second duration. I used a five-minute duration because most of the Operating Systems consider a computer as idle when is inactive for five minutes. This feature is important to prevent the server from making decisions based on temporary changes within the environment.

The server, however, does more than the steps outlined in figure 6. When a card is frequently pointed on the reader, for instance, the server takes into account the first input and ignores the rest. If the existing user in the venue points her card after five or more minutes, the server infers that the user is leaving the room. Before deleting the user from the list of users who are in the room, however, the server monitors the status of her devices for next two minutes. If there is no interaction within that period, the server then deletes the user. Otherwise the server treats the detection as unreliable and ignores it. The best solution, however, would be to use two readers; one positioned near the door and the other a bit further. So when a user enters the room, her card will be detected first by the first reader and vice-versa.

**Algorithm 1** Server Algorithm

```
1   Listen from the proxy computer
2   while (input from proxy computer is not null) do
3     Initialize the reasoning engine
4     Load the knowledge base
5     Read the system time
6     Open user activity logger
7     Open inference activity logger
8     Interpret data from the proxy computer
9       if (the input is from an RFID reader)
10        Get information about the user
11          if (room is empty)
12            go to 1
13          Else
14            while (the room is NOT empty) do
15              Check the status of user's computer
16                  if (computer is ON)
17                    Check user's keyboard activities
18                    Check user's mouse activities
19                  end if
20              Update the user's status
21              Log the user's status
22            end while
23            Insert facts about the room into the engine
24            Match the rules from the knowledge base
25            Fire the matched rule
26            Trigger application manager
27            Log the engine's decision
28          end if
29      elseif (the input is a notification message)
30            go to 11
31      end if
32      if (the user is not in the room)
33          Add the user in the list of the existing users
34          go to 15
35      end if
```

Figure 6. Pseudo code for the server

## 4. Application of KoDA

To evaluate a context-aware architecture, framework or middleware, researchers have been using imaginary or working context-aware applications. Kaenampornpan [18], Kofod-Petersen [19] and Henricksen [6], for instance, used scenarios to illustrate how their solutions can be used in a real-world. Biegel [5], Chen [20] and Dey [7] implemented and used working context-aware applications to demonstrate how their solutions can be used in a real-world. Using scenarios is a feasible approach since the focus of these solutions is not on implementing context-aware applications. Nonetheless, scenarios are far from reality to fully show the potentials of these solutions. Hence, this work adopts both approaches to evaluate KoDA.

### 4.1. Remotely Switching Devices ON/OFF

To illustrate how KoDA works, an application to remotely switch ON or OFF IP-based devices depending on a context was developed. As shown in figure 2, the application, through the application manager, are included in the right-hand side of inference rules and therefore can only be invoked if ongoing context of a user is determined. Upon recognizing a user in a venue, the server takes MAC and IP addresses of the user's devices. For illustration purposes, once the prototype recognizes an ongoing context, it triggers the application to remotely switch ON or OFF the devices.

Assume that this is an application to remotely change the alert mode of the users' mobile phones. If the recognized context is a meeting, for instance, KoDA would trigger this application to seamlessly change the settings of the user's phone from the ringing mode to the silence or vibrating mode. The users would not have to worry about where they enter and what settings their phones have. KoDA would make the use of mobile phone intuitive and thus contributing to the vision of UbiComp.

### 4.2. KoDA with Microsoft Cortana

Microsoft Cortana[4] is an intelligent personal assistant application for Microsoft Smartphones. It combines voice recognition and context-awareness to effortlessly assist a user. One of the boasting feature of Cortana is its ability to automatically transfer phone calls to voicemail when you do not want to be disrupted. This feature is useful, for instance, when you are briefing your boss about a product or giving a keynote speech in a conference. With Cortana installed in your Smartphone, you simply need turn it ON when you do not want to be disrupted and OFF when you are in your normal routines.

Nonetheless, Cortana cannot recognize ongoing context and hence depends on a user to turn the feature ON or OFF. Subsequently, this requires a user to continuously be aware of her social settings and settings of her Smartphone to effectively use this feature. So before meeting your boss for briefing, for instance, you need to remember about the meeting and turn the feature ON. Once you finish the meeting, you need to remember to turn the feature OFF. Using Cortana with KoDA removes the need of the users to continuously remember about their social settings and the settings of their devices. Hence there is a potential for increasing the users' productivity when KoDA is used.

## 5. Conclusion

This paper presents a Knowledge-driven Distributed Architecture (KoDA), describes implementation of a prototype and illustrates how the architecture can be applied and used in a real-world environment. Unlike the existing architectures, KoDA is designed based on a knowledge-intensive model hence enabling it to make its decisions based on a comprehensive model of the real-world environment. Like any other context-aware architectures, KoDA is a 3-layered architecture comprised with perception, inference and application layers. Through the perception layer, KoDA gathers facts of a venue and share with the inference layer. The inference layer uses the facts to determine ongoing context and communicate with application layer that triggers appropriate applications.

To illustrate capabilities of KoDA, a prototype has been developed. To illustrate application and usage of KoDA in a real-world, the prototype was set and used in a real-world environment. KoDA was able to recognize ongoing contexts of users in the environment and trigger an application to perform a task. On another occasion, a scenario of using Microsoft Cortana (i.e. an intelligent personal assistant) with KoDA has been used. The scenario shows that KoDA can work well with Cortana because the users will not have to think about their devices, social settings and whether settings of their devices conform or not. All these will be taken care by KoDA since it can dynamically recognize ongoing contexts and communicate with applications to respond accordingly.

Through its distributed nature, KoDA can be used to support multiple rooms in a workplace. KoDA can also support resource-constrained computing devices. Through its ability to continuously monitor, infer and respond to changes in an environment, KoDA can dynamically recognize ongoing contexts. Furthermore, KoDA is designed with a middleware that separates connectivity processes from interpretation processes. Complemented by a generic and knowledge rich context model, which allows a subset of context parameters to be implemented, KoDA is more scalable and flexible to new technologies as they emerge.

Experiments are crucial in a scientific research in Computer Science and are core to the evaluation of any UbiComp system. Therefore the next step of this work is to perform experiments to measure accuracy of KoDA on recognizing contexts. KoDA is capable of storing information about recognized contexts and evidences used to infer these contexts. This information can be used as another source of knowledge for KoDA. The majority of inference engines are incapable of learning and therefore currently such information is not useful. Therefore, a research on learning mechanisms for context-aware architectures is required to such information useful.

[4] https://www.microsoft.com/en-us/cortana/

 

## 6. References

[1] Evans, D. (2011). The internet of things how the next evolution of the internet is changing everything. Tech. rep., Cisco.

[2] Van Kasteren, T., Noulas, A., Englebienne, G. & Krose, B. (2008). Accurate activity recognition in a home setting. In Proceedings of the 10th international conference on Ubiquitous computing, 19, ACM.

[3] Liu, H. (2010). Biosignal controlled recommendation in entertainment systems. Technische Universiteit Eindhoven, Eindhoven, 1133.

[4] Kukkonen, J., Lagerspetz, E., Nurmi, P. & Andersson, M. (2009). Betelgeuse: A platform for gathering and processing situational data. Pervasive Computing, IEEE, 8, 4956.

[5] Biegel, G. (2005). A Programming Model for Mobile, Context-Aware Applications. Ph.D. thesis, University of Dublin, Trinity College.

[6] Henricksen, K. (2003). A Framework for Context-Aware Pervasive Computing Applications. Ph.D. thesis, School of Information Technology and Electrical Engineering, The University of Queensland.

[7] Dey, A.K. (2000). Providing Architectural Support for Building ContextAware Applications. Ph.D. thesis, Georgia Institute of Technology.

[8] Da, K., Roose, P., Dalmau, M., Nevado, J. & Karchoud, R. (2014). Kali2much: a context middleware for autonomic adaptation-driven platform. In Proceedings of the 1st ACM Workshop on Middleware for Context-Aware Applications in the IoT, 2530, ACM.

[9] Roalter, L., Kranz, M. & Moller, A. (2010). A middleware for intelligent environments and the internet of things. In Ubiquitous Intelligence and Computing, 267281, Springer.

[10] Ranganathan, A. & Campbell, R.H. (2003). A middleware for context aware agents in ubiquitous computing environments. In Middleware 2003, 143161, Springer.

[11] B. Schilit, N. Adams & R. Want (1994). Context-aware computing applications. In Mobile Computing Systems and Applications, 1994. WMCSA 1994. , 85-90.

[12] D. Lupiana (2015), A Knowledge-driven Distributed Architecture for Context-Aware Systems, Ph.D. thesis, School of Computing, Dublin Institute of Technology.

[13] Coutaz, J., Crowley, J.L., Dobson, S. & Garlan, D. (2005). Context is key. Commun. ACM, 48, 4953.

[14] Schmidt, A., Aidoo, K., Takaluoma, A., Tuomela, U., Van Laerhoven, K. & Van de Velde, W. (1999). Advanced interaction in context. In Handheld and ubiquitous computing, 89101, Springer.

[15] Forgy, C. (1979). On the Efficient Implementation of Production Systems. Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University.

[16] Siewiorek, D., Smailagic, A., Furukawa, J., Krause, A., Moraveji, N., Reiger, K., & Wong, F. L. (2003). Sensay: A context-aware mobile phone. In null (p. 248). IEEE.

[17] Harter, A., Hopper, A., Steggles, P., Ward, A. & Webster, P. (2002). The anatomy of a context-aware application. Wireless Networks, 8, 187197.

[18] M. Kaenampornpan (2009). A Context Model, Design Tool and Architecture for Context-Aware Systems Design. Ph.D. thesis, Department of Computer Science, University of Bath.

[19] A.Kofod-Petersen (2007). A Case-Based Approach to Realising Ambient Intelligence among Agents. Ph.D. thesis, Department of Computer and Information Science, Norwegian University of Science and Technology.

[20] H. Chen (2004). An Intelligent Broker Architecture for Pervasive Context Aware Systems. Ph.D. thesis, University of Maryland.

## 7. Acknowledgements

Many thanks to the Government of Tanzania, through the Institute of Finance Management, for funding this research. My sincere gratitude to Fredrick Mtenzi, Brendan O'Shea and Ciaran O'Driscoll for their invaluable contributions throughout this research.

 