On the Road to a Reference Architecture for Pervasive Computing

Osama M. Khaled, Hoda M. Hosny and Mohamed Shalan
The American University in Cairo, Department of Computer Science, New Cairo, Egypt

Keywords: Pervasive Computing, Ubiquitous Computing, Smart Object, Open Source, Reference Architecture.

Abstract: An efficient development strategy for pervasive computing requires that the smart object manufacturers design their devices with profound facilities that can be accessible for developers. In our in-progress research, we present a high level design for smart object essential handlers. This design establishes rules and regulations for the development of pervasive computing in general and promotes for quality in pervasive systems in particular.

1 INTRODUCTION

The Pervasive computing concept was first introduced by Mark Weiser (Weiser, 1991) in 1991 as if he was reading the future of computers in the 21st century. Weiser was convinced that personal computers are not satisfactory enough for integration into humans' lives in a natural way. He was convinced that computation will converge to become ubiquitous. In other words, computation will be present "everywhere" and will be featured by its invisibility to the human eyes, yet available for people to use unconsciously. This vision may have been impossible to achieve during the 90s of the last century, but we do nowadays have all the technologies that we need to make Weiser's vision come true. We have advanced wireless networks distributed in many areas, LTE networks across all countries, hand-held and mobile devices with integrated sensors, appliances with embedded computers and wireless controllers. In addition, industry and universities are more willing to invest funds on research in these areas. MIT, IBM, UC Berkeley research projects are just examples for such enormous research investments (Zhou, et al., 2010). Moreover, smart cities and mobile health merge now to enrich human lives with smart health (Solanas, et al., 2014).

Researchers who work in pervasive computing face many challenges, however. Pervasive computing is a descendant of other computing fields, like distributed systems, and mobile technologies along with their existing challenges. It is characterized by the common appearance of factors like context-awareness, system adaptability, and volatility. In addition to the above, researchers are faced with privacy, security, safety, and limited resources as crucial issues that must be resolved. As sensed from the term ubiquitous, personal information may be collected and distributed without permission from its owner. This can raise legalization issues that must be resolved within the information distribution laws. Also, if security can be breached for devices, appliances, or cars, this can cause high risks to their users, which results in safety concerns that must be handled as well (The Parliamentary Office of Science and Technology, 2006). The challenge of limited resources is inherited from the mobile technology, but it becomes more apparent with pervasive computing since the processing requirements will constantly increase. This can lead also to higher consumption for devices' resources, such as batteries.

There are some fundamental research challenges for pervasive computing systems. They can be listed briefly as follows (Dargie, et al., 2012):

1. Adaptive control: where ubiquitous devices may need to make decisions using uncertain data
2. Reliability and accuracy: where future work needs to address accuracy of recognition algorithms and the possibility of making use of cloud computing resources.
3. Security and Privacy: how a device can recognize other sensing devices and employ proper security and privacy strategies.
4. Hybrid Intelligence: mixture of non-

Khaled O., Hosny H. and Shalan M..
On the Road to a Reference Architecture for Pervasive Computing.
DOI: 10.5220/0005230400980103
In *Proceedings of the 5th International Conference on Pervasive and Embedded Computing and Communication Systems* (PECCS-2015), pages 98-103
ISBN: 978-989-758-084-0

deterministic and deterministic intelligence mechanisms to reason about context types.
5. Unified architecture: where a rapid and common architecture is required.
6. Tool Support: the need is still there to have tools to support rapid development of context-aware systems

The following sections will give an overview for an in-progress research that addresses some of these challenges. Section 2 presents some of the related research work in that field. Section 3 and its subsections give more details about our proposed high level design and section 4 concludes the paper.

2 RELATED WORK

There is a number of recent research efforts in open-platforms for pervasive computing. Check and Kotz (Chen & Kotz, 2002) designed an open event-driven platform called Solar which responds to context changes, represented as events, and interested applications subscribe to event streams, continuous list of events, and these, in turn, react accordingly.

There is also a good research in the Internet of Things (IoT) field as well for building open platforms. Kim and Lee (Kim & Lee, March 2014) developed an Open Source platform called **OpenIoT** in order to recognize an ecosystem that comprises different stakeholders. According to Kim and Lee the **device developer** provides the suitable device to host an application which is generated by a **software developer**. The **service provider** purchases the application and asks the **Platform Operator** to host it. The **Service User** then uses the application using the **Network Operator**. The framework consists of four major platforms (Planet Platform, Mash-up Platform, Store Platform and a Device Platform to facilitate (for the ecosystem stakeholder) the interaction through open-source APIs.

Some researchers believe that in order to facilitate the development of pervasive computing, then an open infrastructure has to be there in their City. Ojala and Kukka (Ojala & Kukka, 2009) promote an open infrastructure project in the city of Oulu, Finland. They provided computing facilities in the city where users can have access to WiFi, Bluetooth, SMTP servers, Large LCD displays equipped with RFID and NFC readers. They present a large-scale infrastructure to make their city ubiquitous.

There are also some open frameworks that target the development of pervasive systems with different capabilities and that are designed for different purposes. For example, the JCAF (Java Context Awareness Framework) (Bardram, 2005) is a java based framework for implementing context-aware applications. The CMF (Context Management Framework) by Korpipää et al (Korpipää, et al., 2003) was designed for Symbian mobile phones. It allows real-time context reasoning for information even if there is noise.

Another related research work by Walach et al. (Walch, et al., 2013) adopts a process-based development approach. The authors developed a tool to capture specifications for home automation and convert them to BPEL (Business Process Execution Language). On the other hand, Oliver and Broadbent (Oliver & Broadbent, 2013) worked on home smart devices as well but to capture their network traffic and profile devices for further analysis. According to the two authors, having a database of network profiles will help in understanding energy consumption at home as well as the smart devices' behaviour in relation with other devices at home.

Some of the aforementioned research efforts would have been more efficient if there were unified architecture for pervasive computing services. Section 3 will explain part of our vision to standardize smart object interface.

3 HIGH LEVEL DESIGN

Figure 1: Smart Object Standardization Handlers.

One solution to address the above mentioned challenges as mentioned earlier in the introduction section, is to standardize a smart object (SO) with handlers that can address key quality issues as shown in Figure 1. These standards can guarantee a controlled open platform that developers can use. The developer need not only know how to program the smart object, in case its interface is available for any programmer, but needs to know also extra

details that are considered essential for robust and safe pervasive systems.

Smart objects could be equipped with sensors, communication interfaces, processing capabilities, and actuators. Some usage scenarios of these objects may put some living creatures' lives at risk (Yang & Helal, 2008). Accordingly, software engineers need to study quality trade-off options very carefully. Hence, We recommend the following standards for smart objects:

1. **Programming Permissions:** as they are objects in a physical world, they will have unique identifiers, and as they may risk lives if not used properly, as well as expose privacy and security of people, the object will have three levels of protections for its programmable interface:
   a. **Public interface:** which can be used by designers without permission from the manufacturer
   b. **Protected Interface:** which can be used by designers who are certified by the manufacturer
   c. **Private Interface:** which can be used only by the manufacturer's engineers.
2. **Safety procedures:** as smart objects co-exist with living creatures including humans, it is essential to know all safety procedures associated with their use. This is not only some documents to read, but it may have an interface to access as well.
3. **Security and privacy procedures:** rules to follow in order to secure data processing by that object and at the same time protect the user's privacy
4. **Volatility status:** the developer should be able to determine the volatility expectations during design and later during run-time. Otherwise, the entire system may fail unexpectedly.
5. **Processing Power status:** Every object should reveal its processing status (processing availability and memory status.
6. **Process Hosting: A** SO should have an easy access to its processing power (processor and memory) if there is enough room and if its operating system allows it.
7. **Community statistics:** these are statistics that the smart object collects about itself and makes available for other developers. This should not reveal any personal information. It will just help developers understand how to deal with different smart objects in different contexts.

A development framework emerges from the above mentioned elements where different stakeholders work together to create a truly smart environment as shown in Figure 2. Manufacturers produce the smart object and facilitate its usage. The developer builds pervasive systems where he/she can use a protected object handler only if he/she is certified for that through trusted organizations. Then smart objects share their run-time business and technical context for the benefit of the developers' community.

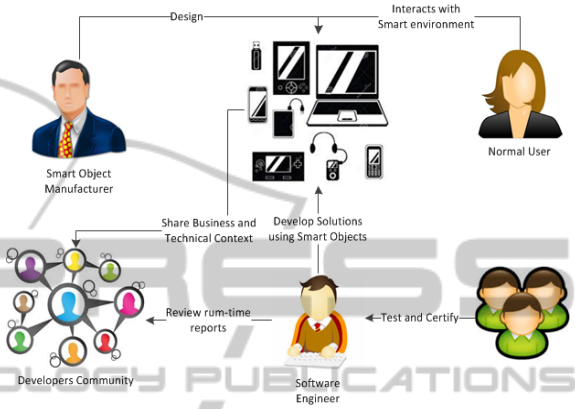

Figure 2: Open Development Framework for Pervasive Systems.

3.1 Programming Permissions

A smart cooker can have different programming methods to allow others to control it. Figure 3 shows a hypothetical cooker class that has some attributes and some methods. The (+) is for public, (-) for private, and (#) is for protected. The semantics here is different from the normal OOP approach, although the same terminologies are used. Accordingly, Height, Width, and IsOvenDoorOpen are public for any developer to use without having permission from the manufacturer.

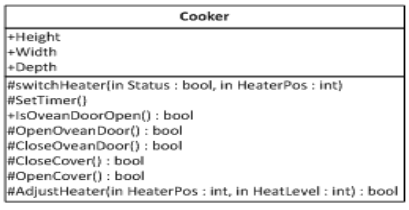

Figure 3: Cooker Handler Class Example.

The main purpose of certifying a software engineer for using the Cooker interface, is not because of the complexity of the object handlers, it is to ensure that the software engineer is capable of designing robust solutions that will not endanger lives. Certification could be standardized via

international organizations that provide recognized certificates world-wide. Certificates can then be implemented as digital certificates signed from one of these trusted organizations and validated by the smart object at run-time. However, the software engineer can still use any of the public and protected handlers during the development phase.

3.2 Safety Procedures

It is important to differentiate between what the designer should do in order to protect the smart object's internal hardware components from damage, and what he/she should do in order to keep the surrounding environment safe. In the first case, the designer is constrained with hardware limitations and he/she should be aware of these before providing any method that can be used by external programmers. In the second case, the designer must assume hypothetical scenarios from real life and modify its design accordingly so that the safety of the smart object is achieved to the best level.

The certified programmer should be able to use protected handlers within the safety procedures provided by the manufacturer. For example, if the door of the smart *Cooker* is open and will risk the safety of close humans while the room temperature is below -20° and there is no temperature sensor attached, then the designer must force the programmer to provide the room temperature before calling the method *OpenOvenDoor(int: Temprature)*. The handler will then give the proper warning in order to check for the proximity of humans before executing the called handler.

It is always safer to equip smart objects with the needed hardware capabilities that allow it to take proper decisions rather than leaving it for the external programmers. However, the cost trade-off is always a factor in the production equation which may require from the designer to design for safety procedures as if insufficient resources are available.

3.3 Security and Privacy Procedures

Security and Privacy is one of the most researched topics in pervasive computing. Security and Privacy of users are combined together as the probability that they affect each other is very high. If system security is breached, then it is possible to release private information about users. On the other hand, if user privacy is violated, it is possible to breach the system using real data which can be used then by the wrong hands and violate the system security.

SO designers should adapt the proper solution to protect customer information and maintain system security. For example, information transferred among smart objects can be encrypted if they release confidential information. Users may need to authenticate their identity during various activities according to the required security level.

Solutions are there and they are straight forward. However, the designer must take his/her decisions wisely since enforcing security rules like encryption may impact the smart object's battery, and hence impact the availability of the environment. Moreover, requiring the users to authenticate constantly may degrade the usability of the solution.

3.4 Volatility Status

SO is volatile if it disappears from the environment without prior alarm. In ubiquitous computation, such behaviour is common rather than exceptional (Coulouris, et al., 2012). Smart Objects can disappear for different reasons, for example:

1. The SO is on the move and its existence in the environment is transient.
2. The SO battery runs out of charge
3. SO hardware failure
4. One of the SO accessible services fails although the SO remains functional with other services.
5. A communication failure impacts the data transformation
6. Network communication bandwidth congestion.

Some of the major issues that may be caused by the SO's sudden disappearance are data corruption and incomplete operations. Technical solutions that deal with hazards like frequent retrials and data hashing can consume substantial traffic and negatively impact the availability of the environment.

One of the essential SO handlers is to inquire about the charging lifetime of the battery. It is important to know this information at run-time since factors like rate of data processing and network communication may change the battery's ideal time-to-charge value.

It can help the solutions designers a lot to take decisions during run-time. For example, the designer may take quick decisions like warning system administrators to charge the SO devices, or switch traffic to standby SO devices.

However, the solution designer should set expectations based on the maximum threshold for battery time-to-charge and use SO battery handler as well to change environmental rules dynamically.

Accordingly, designers can set time constraints rules on some objects, or ensure a higher data protection mode for objects that are about to disappear in order to mitigate the volatility risk.

It is important to mention that the World Wide Web Consortium is drafting a new API document to inquire about the hosting device battery (W3C, 2014). This feature is available also on Andriod platforms for smart phones developers to use it as well (Developers, n.d.).

The purpose is to take informed decisions before the device disappears from the smart environment. A battery is only one reason that can make the SO disappear. The other listed points are also crucial and can greatly affect the availability of the SO. Accordingly, monitoring the congestion of the network packets can give better expectations. The rate of hardware failure, if recorded, can also give a good indication. The proximity of the device from the WiFi hotspot can show real expectations as well. A software bug is another reason that impacts device volatility status.

3.5 Processing Power Status

One of the basic operating system functions is to know the processing power (processor and memory) status. Such knowledge helps in anticipating the environment's availability and time-to-finish for processes. As explained above, an increased processing cycle consumes more power and consequently battery-dependent devices deplete quickly.

The device must give priority for this handler to run as it should normally be called to take a decision based on the device processing power status. However, programmers should be very careful about the frequency of using this method in order not to cause frequent interruption for SO processes and deplete the device battery.

3.6 Process Hosting

Some processes may fail in a smart space if they do not fulfil their tasks. A process may be considered failed if it exhibits one of the following during run-time:

1. The process fulfilled part of its tasks, and failed to complete the remaining tasks
2. The process completed its tasks beyond its service level.
3. The process failed to accomplish all its assigned tasks

One of the main reasons for failure, if faults due to wrong design are ignored, is that the device cannot provide the required resources for the process as needed and on time. In other words, a process may need to have 50% of the CPU processing power to complete its tasks in 1 second as a hard limit for its service level, but because the CPU has other running processes, it succeeds in 1.5 seconds. The failure could be also because the available memory does not satisfy the needs of the process.

The point here is to make use of the environment's ideal resources to support processes that are about to fail in order to sustain a robust smart space. It means that smart objects may **host** processes to make them complete their tasks successfully. The idea of hosting is to help processes recover instead of leaving them fail, if possible, by providing them with needed resources as long as these resources are device-independent and will not harm the SO in an way other than taking more processing power.

3.7 Community Statistics

The development community needs to know more helpful information about different smart objects and their behaviour in different contexts. Context may be understood differently by different people. The business analyst may be more interested in the business context of the SO. The solution architect needs to know the technical context including information about processor, memory, disk storage, sensors, actuators, operating system, network interfaces, temperature, battery, and any other relevant information.

Knowing information about the business context of the SO will help in gaining knowledge about the expected performance of the SO in similar environments. For example, a camera may be exhausted in a prison recording videos and taking snapshots continuously. On the other hand, it may be switched on and off in a school according to school operation times.

Similarly, understanding the technical context of the SO during runtime can help the solution architect decide on the best configuration and design for the SO. For example, if it is reported in the community that the SO temperature increases exponentially when network packets increase by a certain factor, then this causes the device to halt. The designer can then enforce throttling on the network traffic in order to increase the availability of the device.

Private and confidential information should not be shared in the developers' community, and the manufacturers should take care of that. The device

programmer should configure the reporting feature properly and take into consideration the type of network, e.g. whether it is LAN, WAN, or Internet.

If there is a single database about different SOs showing their performance, then data can be analysed easily and a rich set of statistics can be made availed for programmers and designers upon need. Good solutions can be built over a database to avail useful reports and manufacturers as well and other programmers worldwide can benefit from it.

4 CONCLUSIONS

Development methodologies need to evolve differently and quickly in order to cope with the fast growth in technology. Business analysts and architects in particular need to be knowledgeable about different technologies. They need to learn about psychological, sociological, and health precautionary procedures as well. This is essential since smart objects may affect living creatures even if they bring huge benefits to our lives. Hence, designing a smart environment for safety, privacy, and security is mandatory.

Given that there are now a large variety of smart objects with different technologies, software development becomes more complicated too. No one can know everything about all issues and problems related to that domain. Hence, there is a need for an open platform community that shares analytical reports about different smart objects in a systematic and automated manner.

Our target is to have a robust and innovative open reference architecture that helps software engineers working in that field. The reference architecture will capture the state of the art in the domain area of pervasive computing with respect to design patterns, and architecture standards.

REFERENCES

Bardram, J. E., 2005. The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications. *Proceedings of the 3rd International Conference on Pervasive Computing (Pervasive 2005),* May, Volume 3468 of Lecture Notes in Computer Science, pp. 98-115.

Chen, G. & Kotz, D., 2002. Solar: An Open Platform for Context-Aware Mobile Applications. *In Proceedings of the First International Conference on Pervasive Computing (Short paper),* June.pp. 41-47.

Coulouris, G., Dollimore, J., Kindberg, T. & Gordon, B., 2012. *Distributed Systems Concepts and Design.* Fifth Edition ed. s.l.:Addison-Wesley.

Dargie, W., Plosila, J. & De Florio, V., 2012. *Existing challenges and new opportunities in context-aware systems.* New York, NY, USA, ACM, pp. 749-751.

Developers, A., n.d. *BatteryManager.* [Online] Available at: http://developer.android.com/reference/android/os/BatteryManager.html.

Kim, J. & Lee, J.-W., March 2014. *OpenIoT: An open service framework for the Internet of Things.* s.l., Internet of Things (WF-IoT). IEEE World Forum on 2014, pp. 89,93, 6-8.

Korpipää, P. et al., 2003. Managing context information in mobile devices. *IEEE Pervasive Computing,* July-September, 2(3), pp. 42-51.

Ojala, T. & Kukka, H., 2009. *A Digital City Need Open Pervasive Computing Infrastructure.* State College, PA, In Proceedings of Digital Cities 6: Concepts, Methods, and Systems of Urban Informatics.

Oliver, B. & Broadbent, M., 2013. HomeFlow: Inferring Devices Usage with Network Traces. *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication,* pp. 815-820.

Solanas, S. et al., 2014. Smart Health: A Context-Aware Health Paradigm within Smart Cities. *IEEE Communications Magazine*, Aug, Volume 52, no.8, pp. 74-81.

The Parliamentary Office of Science and Technology, 2006. PERVASIVE COMPUTING. *POSTnote,* May.Issue 263.

W3C, 2014. *Battery Status API.* [Online] Available at: http://www.w3.org/TR/battery-status/

Walch, M. et al., 2013. homeBLOX: Making Home Automation Usable. *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication,* pp. 295-298.

Weiser, M., 1991. The Computer for the 21st Century. *Scientific American,* September 1991.

Yang, H.-I. & Helal, A., 2008. *Safety Enhancing Mechanisms for Pervasive Computing Systems in Intelligent Environments.* s.l., Sixth Annual IEEE International Conference, pp. 525-530.

Zhou, J., Gilman, E., Ylianttila, M. & Riekki, J., 2010. *Pervasive Service Computing: Visions and Challenges.* s.l., 10th IEEE International Conference on Computer and Information Technology (CIT 2010), pp. 1335-1339.