PERVCOMPRA-SE: A PERVASIVE COMPUTING REFERENCE ARCHITECTURE FROM A SOFTWARE ENGINEERING PERSPECTIVE

Osama M. Khaled, Hoda M. Hosny, and Mohamed Shalan

Department of Computer Science and Engineering,
The American University in Cairo, Cairo, Egypt

ABSTRACT

Pervasive Computing is a very challenging and complex domain that still lacks a comprehensive unified architecture. In this paper, we propose a reference architecture for pervasive computing that captures most, if not all, of the key challenges and provides a new architecture model that can be used in almost any business context. It provides conceptual views for the smart environment (SE), the smart object (SO), and the pervasive system (PS). We evaluated the model using a simulation prototype to predict its reliability at runtime.

KEYWORDS

Pervasive computing, Ubiquitous computing, Context-aware services, Internet of Things, Software architecture, Reference Architecture, Model-Driven Architecture, Software design

1. INTRODUCTION

Pervasive computing (PervComp) is one of the hottest topics for research nowadays.Its challenges exceed the outdated main frame and client-server computation models. Its systems are characterized as volatile, mobile, and resource-limited. They stream a lot of data from different sensors. In spite of these challenges, a PervComp system should be highly distributed and should receive multiple visits from different users using hybrid smart devices concurrently. Moreover, the system is expected to be sensitive to the environment and to adapt to the changes smartly and spontaneously.

The above implies by default, a lengthy list of quality features like context sensitivity, adaptable behavior, concurrency, service omnipresence, and invisibility. Such features entail additional challenges in the PervComp system like data security, privacy, quality of service, and fault tolerance. Consequently, these challenges require well-designed systems that consider all such features.

Fortunately, the device manufacturers improved their enabling technologies, such as sensors, network bandwidth, and batteries to pave the road for PSs with high capabilities. On the other hand, this domain area has gained an enormous attention from researchers ever since it was introduced in the early 90s of the last century. Innovative systems have been applied in different business contexts such as learning, emergency, retail, and health. Albeit, it is still classified as one of the visionary systems that are expected to be woven into people's daily lives.

A unified architecture is one of the fundamental research challenges for PervComp systems [1] where a rapid and common architecture is much required. Ashraf and Khan [2] reported 26 challenges that are either not addressed or partially addressed. Some key architectural challenges, namely: Software Structuring, Integration, and conceptual modeling are among the top challenges that they found. Gazis et al. highlighted four architectural challenges in the Internet of Things (IoT) domain as well in a recent research paper surveying systems in the USA, Europe and China [3]. They named Reliability, Privacy and Security, interoperability, and device heterogeneity as the key challenges for the successful development of an IoT system.

The initiatives to provide a unified architecture are still very limited and focus on the IoT domain primarily[1]. It is worth mentioning here that there is already an existing RA for the IoT called IoT-A [4] since 2013; however, the IEEE Standards Association started another project to set architectural framework standards for the IoT domain. The project is active and has not been finalized until the writing of this document [5]. These initiatives focus mainly on the IoT, which mandates that objects should be Internet-enabled by definition, while PervComp, which is more generic, can accept objects to be Internet-enabled or not.

Moreover, the purpose of the unified architecture is not only to speed up the development process of a new software product, but more importantly is to bring all the software engineers into a common ground of understanding[6] by generating and sharing the same terminologies. Failing to interpret the different terminologies into common meanings can lead to a project's failure [4].

PervCompRA-SE provides a comprehensive reference architecture (RA) to generate concrete architectures for PervComp that can be adapted in different business contexts. It is a business-driven reference model covering 17 quality features with an extensive study in both the business and technical aspects of the RA. It resulted into practical business and technical models accompanied by guidelines and a trade-off analysis. We evaluated our technical model extensively using qualitative and quantitative methods.

The literature includes definitions for a Practice Reference Architecture (PRA) and a Futuristic Reference Architecture (FRA) [7].A PRA tries to capture best practices from existing architectures along with architectural patterns in order to facilitate the implementation of concrete architectures. Its intent is to resolve time-to-market and standardization problems. On the other hand, a FRA is built to become the first type. It must be based on research and it has to introduce innovative ideas [7]. Once an FRA is implemented as a concrete architecture it becomes an immature PRA, which encourages others to adopt it in more implementations and to transform it finally into a PRA.

----

[1]Some researchers label IoT as a branch from the PervComp and some others use the terminology to refer to the pervasive computing domain.

PervCompRA-SE is an FRA that captures best practices and introduces innovative features as well. The RA that we intend to build will be a visionary architecture. Hence, the focal point that this research addresses may be summarized as follows:

With the fast spread of pervasive systems, is it possible to generate a futuristic reference architecture for pervasive computing systems that encompasses most, if not all, architectural challenges and that can be applied/adapted in different business contexts?

The paper is organized as follows: section II provides our research methodology, section III covers the related RAs that we surveyed, section IV explains the technical reference architecture, section V describes the evaluation methods that we adopted, and section VI concludes the paper.

2. RESEARCH METHODOLOGY

PervCompRA-SE is designed to reflect the best practices in building reference architectures as well as capturing most, if not all, quality features in PervComp. The best practices approach that we adopted [6][8] is that the RA has to:

1. Capture the Essence of Existing Architectures.
2. Has an architectural baseline model.
3. Provides sufficient Guidance.
4. Considers the Business Needs.
5. Considers the Business Context.
6. Provides a Common Dictionary.
7. Captures and Shares Architectural Patterns.
8. Has an Architectural Vision.
9. Has a Prototype.

A PervComp system exhibits, as mentioned by Spínola[9], some key quality features. These are domain independent quality features that we classified into business and architectural quality features based on their proximity from the business and architectural contexts of PervComp.

We gave a clear description for the business quality features in our basic requirements model as explained in [10], which includes Adaptable Behavior (AB), Context Sensitivity (CS), Experience Capture (EC), Fault Tolerance (FT), Heterogeneity of Devices (HD), Invisibility (IN), Privacy and Trust (PT), Quality of Service (QoS), and Service Omnipresence (SO) [9]. We added Security (ST) and Safety (SY) to reach 11 business quality features.

We also adapted the 6 additional architectural quality features mentioned by Spínola[9], shown in Table 1.

We followed the normal software engineering lifecycle in order to collect the requirements, and generate the rest of the artifacts by exploring multiple sources in PervComp, and by using collected knowledge, and results of meetings with experts. We analyzed these requirements to generate additional artifacts (e.g. a business ontology, and quality features weights). We then moved to the next phase (design) in order to generate a technical reference architecture (TRA).

We generated the baseline architecture (BLA) model using the artifacts from the business analysis phase, and the artifacts generated from the design phase. Finally, we used different qualitative and quantitative techniques to evaluate the reference architecture (Fig. 1).

Table 1.Architectural Quality Features

Feature	**Description**
Concurrency (CON)	The system design must ensure proper performance and correct behavior of shared resources under concurrent access from different clients [9].
Function Composition (FCN)	The system must be able to produce new services from existing ones based on their specifications [9].
Openness (OPS)	It is a characteristic of a system which is measured by the number of key published services [9].
Scalability (SCL)	A system is scalable when it keeps operating, with an acceptable degree of efficiency, regardless of the increase in resources and users [9][11].
Service Discovery (SDV)	The system should be able to allocate new services, register them, and facilitate access to them according to the environment [9].
Spontaneous Interoperability (SIP)	The system should be able to associate itself with new partners (e.g. sensors, actuators, or peer systems) normally during operation [9].

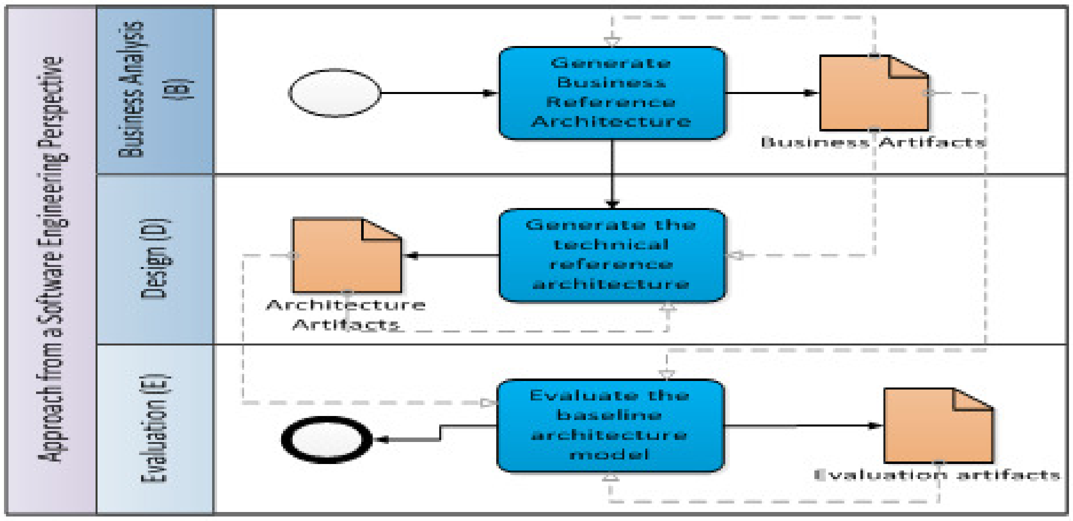
Approach from a Software Engineering Perspective
Business Analysis (B)
Design (D)
Evaluation (E)
Generate Business Reference Architecture
Business Artifacts
Generate the technical reference architecture
Architecture Artifacts
Evaluate the baseline architecture model
Evaluation artifacts

Figure 1. High-Level Approach from a Software Engineering Perspective

We organized the PervCompRA-SE so that the architect or the business analyst can use the business reference architecture (BRA) then proceed with the normal activities to generate a concrete architecture. On the other hand, the architect or the business analyst may proceed to review the TRA then proceed to generate the concrete architecture (Fig. 2). However, it is highly recommended to get acquainted with the concepts and terminologies in the BRA in order to generate a consistent and concrete architecture.

We derived the BLA model from the BRA [10], from the architectural requirements for the quality features shown in Table 1, from network challenges, technology enablers, and from relevant design and architectural patterns.

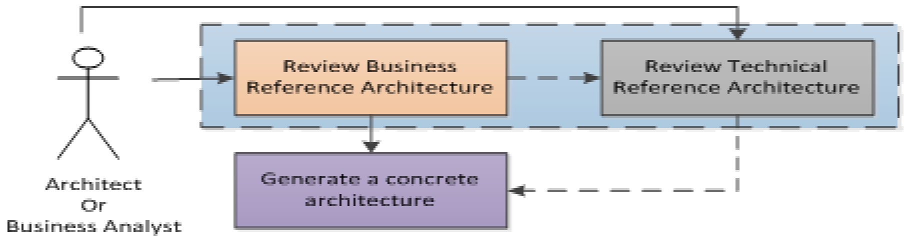
Review Business Reference Architecture
Review Technical Reference Architecture
Generate a concrete architecture
Architect Or Business Analyst

Figure 2. Decoupling BRA from TRA

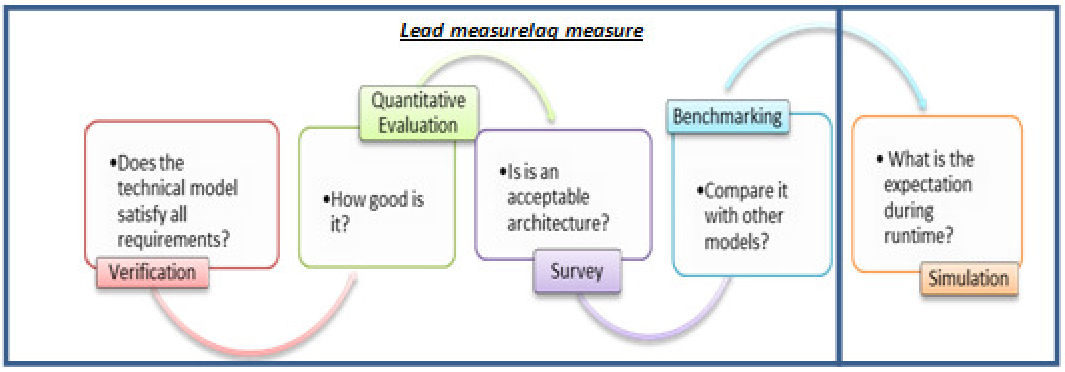
Lead measurelag measure
Quantitative Evaluation
Benchmarking
•Does the technical model satisfy all requirements?
Verification
•How good is it?
•Is is an acceptable architecture?
Survey
•Compare it with other models?
• What is the expectation during runtime?
Simulation

Figure 3. The suggested reference Architecture Evaluation Cycle

We adopted a hybrid approach that combined between qualitative and quantitative techniques. The evaluation cycle, as shown in Fig. 3, aims to trace every module in the BLA to the business and architectural requirements, generate measurements for the architecture metrics, compare these metrics with experts' models generated from the same set of requirements, and verify its acceptance by the development community.

The above evaluation activities give a *lead measure* of the design quality before implementation. They are sufficient for generating a concrete architecture. However, in order to predict its behavior during runtime, there is a need for a *lag measure.* We implemented a simulation experiment to predict the reliability of the PervComp system that adopts the PervCompRA-SE.

Although we provide a lot of abstracted concepts in PervCompRA-SE, we wanted to have special ontological terms that are derived from the quality features and give measurement scales that could be used at runtime. We captured ontological terms from the BRA and TRA and classified them either as *value* or as *issue.* The *value* is a benefit that system users need to gain from the system. The *issue* is a problem or a non-desired aspect that the system users are not willing to have [12].

3. RELATED WORK

There are very few research efforts which position themselves as RAs for the PervComp domain. Hence, we explored RAs from IoT as well because it comes very close to PervComp. We did not include early contributions in the PervComp domain which mainly focused on providing programming frameworks or middleware solutions (e.g. AURA, Gaia, SOCAM, CARISMA, CORTEX, and RCSM) [13][14] as they were irrelevant to our scope.

Table 2.Comparing Related Reference Architectures with Respect to quality features

RA / FT	SO	IN	CS	AB	EC	SDV	FCN	SIP	HD	FT	ST	OPS	CON	QoS	SCL	PT	Score
(1)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓		✓		12
(2)	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓			✓	✓	12
(3)	✓		✓		✓	✓		✓	✓		✓					✓	8
(4)	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	15
(5)			✓	✓	✓					✓	✓			✓			6
(6)	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓	✓	✓	✓	13
(7)			✓	✓						✓			✓	✓			5
(8)			✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	14
(9)	✓									✓		✓	✓	✓	✓		6
(10)			✓							✓	✓					✓	4
(11)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		15

We reviewed a number of related systems, namely: 1) I-Centric [15], 2) PCA_A [16], 3) Self-Care Infrastructures [17], 4) PSC-RM [18], 5) Smart Environment Software Reference Architecture [19], 6) the NGSON Multiplane Framework [20], 7) Component-based Self-Adaptive [21], 8) IoT-A [22], 9) CIPS [23], 10) IoT Security and Privacy [24], and 11) RA-Ubi[25]. Our revision aimed to speculate the level of satisfiability of the quality features and the abidance of the best practices approach as described in our approach (section II).

Most of the aforementioned RA's anchored on specific perspectives of PervComp architecture. For example, the Self-Care Infrastructures RA offered a RA suitable for a pervasive health environment. CIPS showed a RA for highly intensive data processing systems. NGSON highlighted a RA that network operators could adopt in order to provide PervComp solutions. A few of them tried to give generic views that could fit for any solution like RA-Ubi, IoT-A, PCA and I-Centric. Some others just focused on one architectural layer or component.

Tables 2 and 3 compare these RAs with respect to the quality features and the best practices approach, respectively as mentioned above, in section II.

We checked how many of the quality features (in Table 1), except safety, were fulfilled by these RAs as shown in Table 2. It is important to note that some RAs had a specific focus like the RA in (Security and Privacy in IoT) [21] which focused mainly on security and privacy. Other RAs focused on Environment Intelligence [19], and some others were oriented towards the pervasive services infrastructure [20]. The traced features as shown in Table 2 show the following:-

1. The IoT-A, PSC-RM and RA-Ubi considered most of the quality features essential for PervComp systems but the IoT Security and Privacy RA were the least to consider these features.
2. The quality features that the RAs considered most are context-Sensitivity followed by Service Security, adaptable behavior, and fault tolerance.
3. Function Composition and Openness were the least considered quality features.

We note from Table 3 that most of the RAs follow the best practices approaches. On the other hand the least adopted practices were: providing guidance to instantiate a new architecture or are based on a business context. Finally, all of them captured the essence of the existing architectures and were able to provide a common dictionary. This may simply mean that the authors were more concerned with explaining their concepts and making them clear for the readers.

The number of RAs focusing on PervComp is still limited and very few of them follow the best practices guidelines, as mentioned in [6] and [8], to build a robust RA and to cover most of the business challenges. Most of these RAs are not mature enough to provide enough guidance for software engineers and did not consider trade-offs among the quality features.

Table 3.Comparing Related Reference Architectures with Respect to Best Practices Approach

QC / RA	Essence of Existing Architectures	Baseline Model	Guidance	Business Needs	Business Context	Dictionary	Patterns	Vision	Prototyping
(1)	✓			✓		✓	✓	✓	
(2)	✓	✓				✓	✓	✓	
(3)	✓	✓		✓	✓	✓			✓
(4)	✓	✓		✓	✓	✓	✓	✓	
(5)	✓		✓	✓	✓	✓			✓
(6)	✓	✓		✓		✓	✓	✓	✓
(7)	✓	✓	✓			✓	✓		✓
(8)	✓	✓	✓	✓	✓	✓	✓	✓	✓
(9)	✓	✓	✓	✓	✓	✓	✓		✓
(10)	✓			✓	✓	✓	✓	✓	✓
(11)	✓	✓		✓		✓	✓	✓	

4. THE TECHNICAL REFERENCE ARCHITECTURE

We The TRA is explored from technological, network, and design decisions [12] which resulted into a baseline architectural model describing the structure and behavior models as will be explained in the coming sections. The BLA model provides essential details about:

1. **The Smart Environment:** a conceptual view of the SE and classification of the objects.

2. **The Smart Object:** an abstracted view of the SO and the essential handlers that it should include to interact with the SE.

3. **The Pervasive System:** The essential modules that should exist in a PS with high level linkage among them.

4. **The System Optimization:** The basic optimization parameters in the system.

5. **The Architecture Variability:** the essential configurations of the PervCompRA-SE to generate different architectural models based on the changing rules.

6. **The System Deployment:**The essential deployment strategies that could be implemented for a PS in order to increase its reliability.

4.1 Smart Environment

The SE is just an instantiation of the PervComp system where objects show a high degree of intelligence. Ideal SOs possess processing powers (memory & processor), a communication interface, sensors and actuators. According to Kortuem et al. the degrees of smartness could be there among objects based on the manufacturers' designs. Such degrees are categorized into three types [26]. Each type has its associated set of functions, rules, and workflows:

1. **Activity-aware object**: this is an object that can record information about the surrounding activities and aggregates them, but does not respond to these activities.

2. **Policy-aware object**: this is an object that can recognize surrounding activities according to pre-defined policies and devises proper actions and hence can respond by a warning or an alert.

3. **Process–aware object**: this is an object that recognizes surrounding activities in the light of organizational processes and provides proper directions for users about tasks, deadlines, and decisions.

It is important to note here that the SE can be composed of other passive objects that are not smart by design such as RFID-tagged devices which can be identified only by other sensor-enabled objects, which could be SOs as defined earlier. For example, tracking boxes of products coming in/out of a specific warehouse does not require intelligence in these boxes. They just need a reader and RFID stamp-tags per box.

Hence, we reached a generic model for the SE, which is ideally represented as a PS as shown in Fig. 4. The SE is structured as follows:

1. The SE can have a nested SE. Every SE is composed of objects.

2. An object could be a SO or a dummy object. The details of the SO are derived from Microcontrollers and Smart Phones.

3. A SO is classified as Activity-Aware, Policy-Aware, or Process-Aware. It can contain

dummy objects.

4. A dummy object is an object that lacks one of the properties of the SO. It has a specific job responsibility with no intelligence or logic. A dummy object is either an active object or a passive object as explained above.

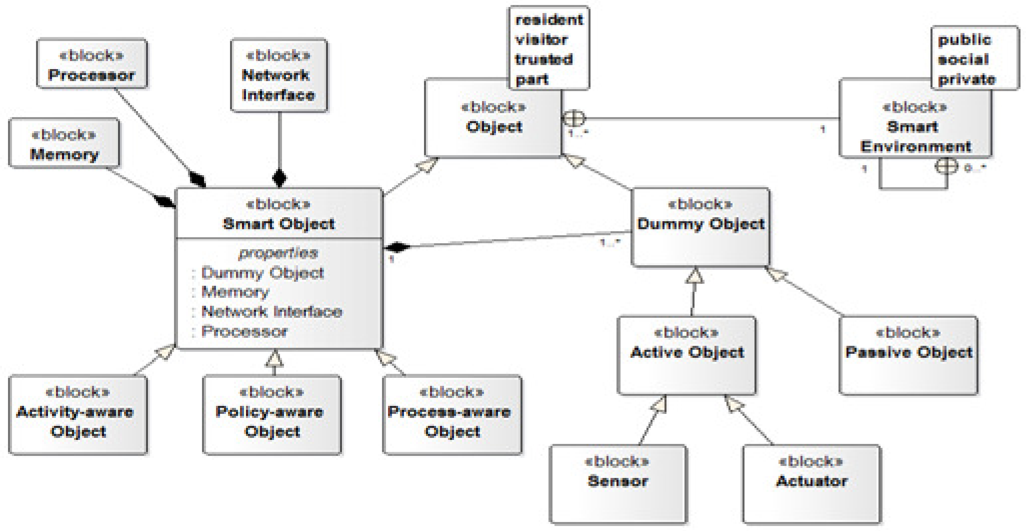

Figure 4. Pervasive Computing Analysis Approach

As shown in Fig. 4 also, a SO must possess some properties, or capabilities, namely processor, memory, network interface, and some sensing or actuation capabilities. We defined some types for the object and the SE, which helps the architect to take better decisions. The SE is an environment that exhibits intelligence behavior through SO(s) that are part object(s) or resident object(s). The SE can be classified from a privacy point of view into [27]:

1. **Public**: where most of its services and resources are accessible to its objects with no access rules.

2. **Social**: that is an environment that grants access to its resources and services based on group association.

3. **Private**: the resources and services are accessible to objects that have the proper permissions for them only.

An object is anything in the world which can be represented in the SE. A classification of the objects based on their interaction model with the SE could be as follows:

1. **Part Object**: an object which cannot be removed from the system, else the system will not function as designed.

2. **Resident Object**: an object which is important as it accomplishes one or more tasks of the system, but removing it will not hinder the system design.

3. **Trusted Object**: an object that the system trusts and that joins the environment frequently.

4. **Visitor Object**: a non-trusted object that joins the environment in ad-hoc situations

All types of objects that join the SE need to interact with the environment in the most optimum way. Hence, there are two types of configuration approaches that can be adopted [11]:

1. **Preconfigured**: the object is bound to the environment through a configuration that aims to establish a long-term relationship between the object and the environment. It applies mostly to the part objects and may be applied to the resident objects.

2. **Spontaneous**: the object is bound to the environment through a spontaneous configuration. This type of configuration applies to the visitor, resident, and trusted objects. The spontaneous binding requires from the system that it negotiates first with the device using a standard protocol, then the system binds it, then the object starts interaction using the proper protocol which was agreed upon during the negotiation step.

4.2 Smart Object

The SO is an important part of a successful PS. It can be programmed to provide the required behavior and can carry out different roles in the SE. Accordingly, we recommend standardizing the SO with handlers that can address key issues. These handlers can add more controls on the PS.

The developer needs not only know how to program the SO, in case its interface is available for any programmer, but also needs to know extra details that are considered essential for robust and safe PSs. Moreover, the final architecture mainly depends on the capabilities of the devices that compose the skeleton of the system. Some usage scenarios of SOs may put some living creatures' lives at risk [28]. Hence, we recommend the following standards for SOs, which we described in details in [29]:

1. **Programming Permissions:** objects in a physical world, may risk lives if not used properly, as well as impact privacy and security of people. The object should hence provide three protection levels for its programmable interface :
   a. **Public:** the API is accessible for the designers without permission from the manufacturer.
   b. **Protected:** the API is accessible for the certified designers by the manufacturer.
   c. **Private:** the API is accessible only by the manufacturer's engineers.
2. **Safety procedures:** the SO must supply all safety procedures either as APIs, configuration, or documentation to read in order to avoid risking the context in which it runs.
3. **Security and privacy procedures:** the SO should provide sufficient APIs that guarantee data security and the protection of the confidential data.

4. **Volatility status:** the SO should provide APIs to determine its volatility status and help in predicting its disappearance from the environment.
5. **Processing Power status:** the SO should provide APIs to reveal its processing availability and memory status.
6. **Process Hosting:** A SO should cooperate with its environment and provide hosting capabilities to execute tasks if there is room for it (e.g. CPU processing is 1% and there is enough memory and storage).
7. **Community statistics:** the SO should provide APIs to gather statistics about it in order to share with the development communities. APIs should not reveal any confidential data. It will help software engineers understand how to deal with different SOs in different environments.

The SO can run in different modes:

1. **Runtime**: where all handlers run with full capacity and with minimal overhead.
2. **Diagnostics**: the SO adds extra overhead to its handlers, like logging, memory dump, etc.
3. **Maintenance**: the SO is in maintenance mode, which means that some of its functions may not be available. For example, its network interface may be disabled, or the handlers that will be disabled will notify the callers that it is in maintenance mode.

4.3 Pervasive System

The PS's behavior is centered on a basic workflow (Fig. 5), by which input devices feed in the system with an event and the system gathers events as context, interprets the context, and then links the interpretation to a decision. The decision, may or may not, trigger actions. These actions are made with output devices, which in turn feedback the system with their results as input data. The workflow cycle is inspired from the human perception process described in [10].

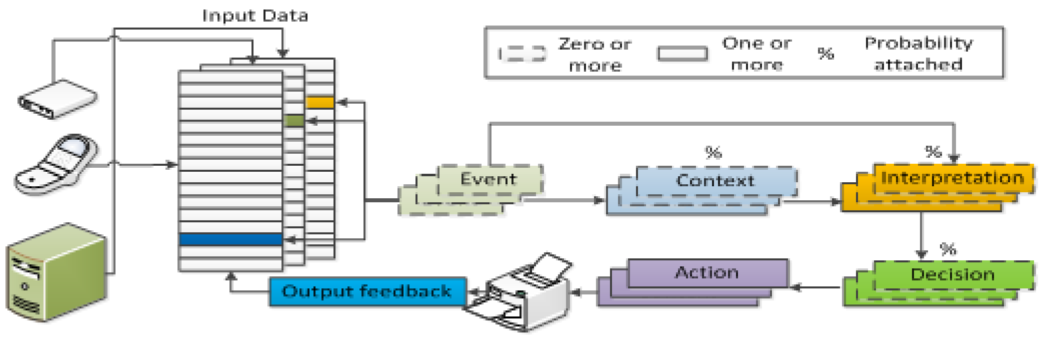

Figure 5. Basic operation workflow in a pervasive system

The PS should consider the uncertainty of the context, interpretation, and decision since the event could lead into different contexts, different interpretations, and different decisions. The basic behavior shows important concepts to understand the whole workflow.

The cycle starts with an **Input** which is a device capable of sending data to the system. The input devices can be classified into **Explicit** and **Implicit** inputs**.** **Explicit** is an input device that feeds the system with external data and requires direct interaction with the system (e.g. keyboard and mouse). **An implicit** input device is a device that feeds the system with external data by detecting the data from the environment (e.g. sensors). The sensor could be a physical or a virtual sensor. It could be a dummy object or a part object in a SO.

A virtual sensor is a software sensor that reads data from other software systems. An example of a virtual sensor is the social network sensor, which reads the status of the user all the time and sends it as an input to the system. A **physical sensor** is a physical device that reads environment conditions like heat, pressure, and light sensors.

The system fetches an **event** which is the basic incident that stimulates the system. The event is identified based on sensed data from different physical and virtual sensors. After that the system identifies a **context** whichis a specific status of the system identified by a set of parameters, a sequence of one or more events. The event can give an indication for one or more contexts; each one may have a different *occurrence weight.* It is described as c = $(e_1, e_2, \ldots, e_n)$ as the system determines the context by detecting a finite sequence of events from 1to *n*. It is important to note that actions of the system may trigger new events that subsequently may lead to new changes in the context.

Interpretation is the logical meaning of the context**.** One or more contexts could have the same interpretation, or the context could have more than one possible interpretation. The events and the context determine the right interpretation. Different interpretations could have different weights as well. The **interpretation** may lead to a **decision** that can be taken. There could be more than one decision, each one with a different weight. The **decision** can lead to zero or more **actions** which are the results of the system **decision.**

The action can be either **visible** or **invisible.** An **invisible** action is taken by the system within its components and does not require direct attention from the users. For example, a self-maintenance action to reallocate resources or free memory is considered an invisible action. The user may review this action later on from the system logs. A **visible** action requires direct attention from the user. For example, a warning message displayed on a screen is considered a visible action. Opening a door as the user steps forward is considered a visible action. A visible action can be further classified as **silent** and **interactive**.

A **silent** action does not require a reaction from the user, e.g. the message or video displayed on a screen. An **interactive** action requires a reaction from the user. For example, switching the light due to opening the room door is an interactive action. Acknowledgment of receiving a warning message is an interactive action.

The system takes actions through an **output** which is a mechanism of the system to make actions. It sends its **output feedback** as the result of the output device, back to the system as an input data.

The whole system may run in different modes. A mode is a special status of the system where operations may have different inputs, execution scenarios, and different outputs. However, modes, in general, will run the system as in its basic operations. The PS should have the following basic modes:

1. **Runtime:** the system runs all its operations in the optimum way.
2. **Assertion:** this is an administrative mode, where details of the system activities are revealed only to the administrator and logged for further analysis.

3. **Out of Service:** this mode should be used if the system should not be shut down and at the same time receives requests but without processing them.
4. **Upgrade:** the system is under upgrade operation which makes one or more modules unavailable until the upgrade process is complete.

Advanced modes could be added to the system to test the results of specific inputs, and outputs or to teach the system and let it improve its rules:

5. **Simulation:** this mode imitates the real world by running hypothetical scenarios over time as if it is running in the real world [30][31].

6. **Teaching:** The system will be in this mode if a lot of details are required in order to feedback the system to improve its artificial intelligence rules [30]

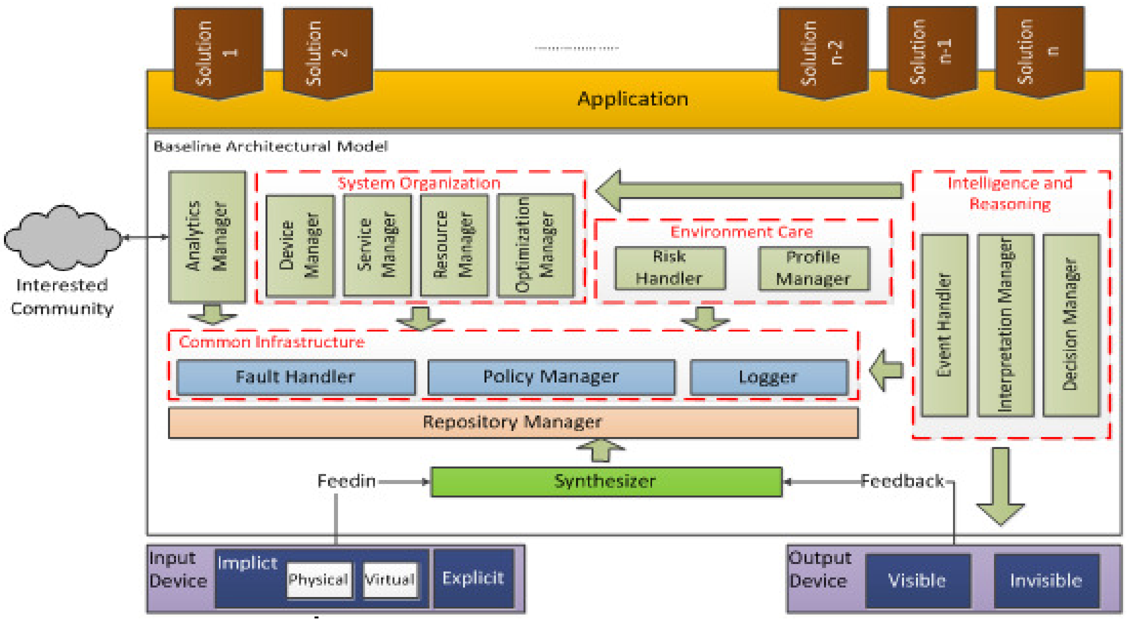

Figure 6. Pervasive System baseline architecture

Fig. 6 shows the structure of the baseline architectural model that we propose for a PervComp domain.

Externally from the architectural view, there is an **application** that implements the functional and quality requirements of the system. Some of these functional requirements are implemented as *solutions*. There could be one or more **solutions** for specific problems installed in the system as plugins. They can be installed/uninstalled in a systematic way. The solution may interact with the core modules of the system.

The **Repository Manager** is the place where data, information, knowledge, and wisdom are stored. The *Repository Manager* is responsible for coordinating the repository operations. The **Synthesizer,** on the other hand is responsible for receiving input from input devices and feedback from output devices, validating them, correcting them if required, and then saving them in the repository. The *Synthesizer* can be part of a middleware and it could also be a built-in service provided by the manufacturer of the input or output devices.

There are **common infrastructure** layers that serve most of the system modules. It is recommended to interact with them *asynchronously* in order not to impact the overall performance:

1) The **Logger** is used to log events, capture system performance in log files which could be in different formats.

2) **Policy Manager:** The policy manager is responsible for managing the system policies and the pre-defined configuration parameters.

3) The **Fault Handler** is responsible for handling all types of faults and taking the proper actions based on the system design as described in [10]. The *Fault Handler* cooperates with the *Interpretation Manager* and the *Decision Manager* to improve its performance.

The **intelligence and reasoning** layer, as shown in Fig. 5, is responsible for handling the basic workflow. Its main components are:

1) **An Event Handler:** is responsible for detecting the events and transforming them into contexts, or linking them with decisions. The *Event Handler* can be part of a **middleware**. The system can assign all middleware responsibilities to it since it is the main interaction point with the rest of the modules. The *Event Handler* may delegate the event to one of the system modules to handle or ignore it if it is already defined to be handled by other modules.

2) **An Interpretation Manager:** The *Interpretation Manager* is responsible for analyzing the interpretation rules and finding new correlations leading it to enhance its reasoning.

3) **A Decision Manager:** The D*ecision Manager* is responsible for analyzing the set of decision rules which are combinations between contexts and decisions with specific weights.

The **system organization** layer is responsible for managing system devices, services, and resources:

1) The **Device Manager** is responsible for registering and tracking all the devices that interact with the system with enough details like (device name, version, manufacturer, manufacturing date, OS version, binding date, last interaction date, unbinding date, display dimension, battery lifetime, etc ...).

2) The **Resource Manager** is responsible for registering the system resources, their locations, their availability per time unit, and their allocation with other objects.

3) The **Service Manager**, which can be part of a middleware, is responsible for registering new services, binding, unbinding, handing over services for mobile users, and producing new composite services.

4) The **Optimization Manager** is responsible for optimizing different system components for the best utilization of services and resources. It is concerned with optimization for quality attributes like processing time, availability, scalability, and responsiveness with respect to the functionalities of the system.

The **Environment Care** layer contains the modules that manage the profiles of the users and risk issues:

1) The **Profile Manager** is responsible for maintaining users' profiles including their preferences and tracking their activities and recording their behavioral trends.

2) The **Risk Handler** is responsible for handling, analyzing, and taking counter measure actions regarding all events concerning highly protected zones (Security, Safety, and Privacy & Trust issues).

The remaining module, the **Analytics Manager,** is responsible for preparing the required statistics about the system. For example, it can aggregate data collected by the *Logger* to show the system performance with different static quality features, like *Context Sensitivity* or *Scalability,* or runtime quality features, e.g. network throughput or reliability. It is responsible also for generating information and knowledge about the system. Some of these statistics will be shared with the interested communities through services published by the *Service Manager*. The **interested Community** is a cloud or a system with details about the usage of devices in different environments.

4.4 System Optimization

The PS's behavior is in continuous change. The behavior may not be correct all the time and accordingly, it needs to be continuously optimized. The optimization process is conducted to assign proper weights for different factors inside the system with predefined standard deviation for every factor. These factors control the behavior of the system and make its choices more accurate. It can be described as a function of a 5-tuple *(Q, C, I, D, S)*:

- $Q = \{q_i \mid i = 1,2,\ldots,n\}$ is a finite set of weights for quality features.
- $C = \{c_i \mid i = 1,2, \ldots, m\}$ is a finite set of weights for different contexts that the system may be in at any point of time
- $I = \{i_j \mid j = 1,2, \ldots, r\}$ is a finite set of weights for different interpretations that the system may use to interpret its current situation.
- $D = \{d_i \mid i = 1,2, \ldots, k\}$ is a finite set of weights for different decisions that the system may take to adapt itself to the change in the context.
- $S = \{s_i \mid i = 1,2, \ldots, h\}$ is a finite set of weights for different solutions that the system may use to implement the adaptation decision.

4.5 Architecture Variability

The *Policy Manager*is responsible for enforcing guiding behaviors on the system. It encompasses the variable behavior of the system during runtime. It is possible to provide different preplanned settings for the *Policy Manager*. The setting may enable/disable some components or features in the system to provide a required behavior. This is not an *Adaptable behavior*, as the adaptability of system will work within the guidelines of the selected policy. We can have what is called a *Dynamic Architecture* of the system. The *Dynamic Architecture* may be defined by the configuration settings in the *Policy Manager.* All these settings manage the component behavior or its relationships with other components [32]:

1. **Enable/Disable Solutions:** solutions could be installed in the system as plugins. There could be different plugins providing similar services but with different functionalities. The *Policy Manager* may choose a policy file with a specific set of solutions based on the mode and the context.

2. **Roles and Responsibilities:** a policy may define a different role and responsibility for a specific object, which is ideally a smart device or a server.

3. **Service product line workflow:** organizes the services or functions, as requested by the *Compose Functions* architectural quality feature.

An architect willing to produce an architecture in a *Product Line Architecture* model may follow the following approach:

- The architect may set the weight for every quality feature or use the default ones as explained in [10].
- Design all possible solutions to resolve the functional and quality problems.
- Set a weighted score for the solutions based on their positive and negative impacts on quality features as shown in our work [33].
- Choose the solutions with the highest scores to produce the design.

The *Product Line Architecture* may change the weights of the quality features and subsequently the solution weights may change, which may lead to a different architecture model.

4.6 System Deployment

There are 3 basic roles that any module/component in the system should be able to assume:

- **Client**: the component requests services from other components.
- **Server**: the component offers services for other components.
- **Peer**: the component requests and offers services.

The classification of the objects in the SE, as shown in Fig. 4 imposes standard preference. An object which is part of the PS should ideally cooperate with other objects in the system to provide the required services for their clients which are usually trusted or visitor objects. However, the system may include some resident objects that can offer services as well, although they can request services from the system as clients. The trusted or visitor objects can act as peers and request/offer services from/to each other if the system is P2P. The sensor offers services for the system since it collects data from the environment. Clients can pull sensor's data upon need. The actuator is similar to the sensor, but it offers actions. In summary, the level of responsibility of the object to produce or consume a service controls the role of the object as client, server, or peer as shown in Table 4.

Table 4.Objects in a Smart Environment and Expected Roles

	Part	**Resident**	**Trusted**	**Visitor**	**Sensor**	**Actuator**
Client	Low	Med	High	High	Low	Low
Server	High	High	Low	Low	High	High
Peer	Med	Med	High	High	Low	Low

5. EVALUATION

As PSs are considered complex, they cannot follow the traditional development cycle. It is recommended to test them first using simulation approaches [34]. We designed a discrete-event simulation experiment in order to predict the reliability of the BLA at runtime. A reliable system is a system that can perform its assigned functionality with a high probability of success during a specified period of time and within specific design constraints [35]. It increases the confidence in the design and gives an early indication for the expected behavior of the system.

We implemented a scenario for tracking a bus equipped with speed and location sensors that travels from point A to point B. The system receives visits from users and optimizes its modules using shared resources as it tracks the trip. The system runs in different modes as well.

We devised a conceptual model towards the complete implementation. Our model is derived from the PervCompRA-SE smart environment conceptual model as described above, in section IV. The model is composed of Entities classified as part objects, resident objects, and visitor objects. The part objects are those modules that define the baseline architecture as shown in fig. 6. The sensors and actuators are resident objects that the part modules interact with to receive data and output data. The smart objects are visitors that request services from the system on regular basis (fig. 7). *Entities* interact with each other through their input and output ports. Every simulation module has two basic attributes *phase* and *tick*. *Phase* represents the status of the entity and *tick* is the logical time by which the entity can accept inputs and generate outputs.

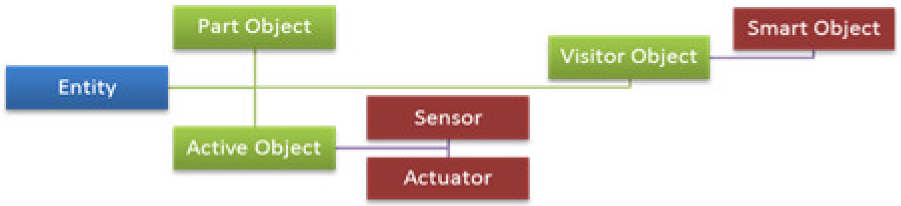

Figure 7. The Simulation Conceptual Model

The whole simulation model can be working in Runtime, Assertion, Security Threat, or Out of Service modes. First, the Runtime mode is the normal execution scenario without changes in the settings of the entities. Second, the Assertion mode is the normal execution scenario but with additional logging activities from these entities. Third, the Security Threat mode is where the whole system is threatened and needs to take some measurements to protect itself. It rejects visits from new smart objects recognized as visitors and accepts visits from the trusted smart objects only. It disables the Synthesizers so that no data can be collected from the sensors. Finally, the Out of Service mode is where the system will not be processing sensor signals, will not accept visits, and will not fetch user profiles from the Repository Manager. The system will still keep recording sensor data and when the system returns to one of the other three modes, then it can fetch the data and continue processing.

Pervasive systems, or IoT systems, are highly vulnerable to security threats [36]. Moreover, systems usually go out of service due to planned maintenance or unplanned outages. It is noticed that the cost of maintaining a system is in continuous increase since the end of the last century. This is basically due to the increased number of developed software applications and their increased complexity [37]. Moreover, administrators dump logs from the system for monitoring purposes all the time. Accordingly, we assumed that the system will be running in normal mode most of the time (64-70%) and there is 30-36% probability that it will be running in one of the other abnormal modes (Assertion, Out of Service, or Security Threat).

We derived other probabilities from different sources to prepare our conceptual model for the simulation with respect to failures of the speed and location sensors, hardware failures, SMS engine reliability, digital screen failure, battery recharge threshold, and rate of accidents. We also assumed that the complexity degree of the module (part object), as shown in equation 1, impacts the probability of failure as well as the probability of repair [38][39].

$$weight = Round(\frac{r * d}{\sum_{i=0}^{15} r_i * d_i} * 100) \qquad (1)$$

Where *r* is the number of satisfied requirements by the part object and *d* is the number of input and output dependency relationships for the part object.

We devised 6 different scenarios with different configurations and different control variables. The configurations were related to the fault handling, optimization, and execution modes. The control variables were divided into best, average, and worse values. We implemented the simulation project using DEV-Suite [40] and Java 6. We executed a total of 45 runs in order to increase the accuracy of prediction.

We captured the Mean Time Between Failures (MTBF) and Mean Time To Repair (MTTR) [41] as per equation 2. A reliability measurement is a function in MTBF and gives a score between 0 and 1 [39].

$$Reliability = \frac{MTBF}{MTBF+1} Availability = \frac{MTBF}{MTBF+MTTR} \qquad (2)$$

The experimental results showed that the model has in the worst cases a reliability of 96.86% and availability of 90.89%. In the average cases, the reliability of the system is 98.08% and availability is 95.77%. In the best cases the reliability is 99.9% and availability is 99.79%.

We also predict an average of 2% additional time needed from the last sensor input calculated against the perfect scenario. The results show that the resource optimization technique that we adopted is working reasonably as it saved 3% of the potential failures of the modules. It was evident that the assumptions for the control variables dominated the general performance of the systems. In general, the scenarios show that the processing time increases as the working conditions get worse.

Although the analysis shows positive results about the reliability and the availability of the architecture model, the prediction is an initial estimation which will definitely change in a real environment. There should be a continuous improvement process by fetching real numbers about the systems' performance during runtime in order to give more accurate predictions about the system failure.

6. CONCLUSION

The pervasive domain is a very complex domain and by providing a unified architecture, standard and robust systems can be more easily implemented. We provided our vision for a reference architecture in the PervComp domain. This vision is based on a detailed study of the business requirements of the PervComp domain as well as the architectural challenges that may be encountered. We provided the details of the simulation prototype which we used to predict the quality of the PervCompRA-SE.

The PervCompRA-SE is a business-driven, safety-aware, open, simple, and almost complete reference architecture. Its design is derived from an extensive study of different business areas and quality features for the PervComp domain. It is one of the few RAs in this domain to handle safety issues. It is not a complex model and its concepts are easy to understand. It addresses 17 quality features through a process that abides by the best practices which makes it offer one of the best choices in this domain for implementation.

We plan to report about the remaining contributions and give more details about the architectural challenges, about the evaluation approaches, and other details of the TRA. It is quite a challenging research area and we plan to continue our work in software product line driven from our reference architecture. We aim to have a large-scale implementation using our architectural reference model as well.

REFERENCES

[1] W. Dargie, J. Plosila, and V. De Florio, “Existing Challenges and New Opportunities in Context-aware Systems,” in Proceedings of the 2012 ACM Conference on Ubiquitous Computing, New York, NY, USA, 2012, pp. 749–751.

[2] M. U. Ashraf and N. A. Khan, “Software Engineering Challenges for Ubiquitous Computing in Various Applications,” in 2013 11th International Conference on Frontiers of Information Technology, 2013, pp. 78–82.

[3] V. Gaziset al., “Short Paper: IoT: Challenges, projects, architectures,” in 2015 18th International Conference on Intelligence in Next Generation Networks, 2015, pp. 145–147.

[4] “Internet of Things - Architecture IoT-A. Deliverable D1.5 – Final architecture reference model for the IoT v3.0.” European Lighthouse Integrated Project, Jul-2013.

[5] IEEE Standards Association, “IEEE Project (P2413) - Standard for an Architectural Framework for the Internet of Things (IoT),” Dec-2016. [Online]. Available: http://standards.ieee.org/develop/project/2413.html. [Accessed: 21-Apr-2017].

[6] R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole, and M. Bone, “The Concept of Reference Architectures,” Syst. Eng., vol. 13, no. 1, pp. 14–27, 2010.

[7] S. Angelov, J. J. M. Trienekens, and P. Grefen, “Towards a Method for the Evaluation of Reference Architectures: Experiences from a Case,” in Software Architecture: Second European Conference, ECSA 2008 Paphos, Cyprus, September 29-October 1, 2008 Proceedings, R. Morrison, D. Balasubramaniam, and K. Falkner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 225–240.

[8] E. Y. Nakagawa, “Reference Architectures and Variability: Current Status and Future Perspectives,” in Proceedings of the WICSA/ECSA 2012 Companion Volume, New York, NY, USA, 2012, pp. 159–162.

[9] R. O. Spínola and G. H. Travassos, “Towards a framework to characterize ubiquitous software projects,” Inf. Softw. Technol., vol. 54, no. 7, pp. 759–785, 2012.

[10] O. M. Khaled, H. M. Hosny, and M. Shalan, “A Pervasive Computing Business Reference Architecture: The Basic Requirements Model,” Int. J. Softw. Eng. IJSE, vol. 10, no. 1, pp. 17–46, Jan. 2017.

[11] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, Distributed Systems: Concepts and Design, 5th ed. USA: Addison-Wesley Publishing Company, 2011.

[12] O. M. Khaled, “Pervasive Computing Reference Architecture from a Software Engineering Perspective,” Ph.D. Dissertation, The American University in Cairo, Cairo, Egypt, 2017.

[13] H. Vahdat-Nejad, “Context-Aware Middleware: A Review,” in Context in Computing, New York, 2014, pp. 83–96.

[14] D. Romero, “Context-Aware Middleware: An overview,” Paradig. Oscar Gonzalez, vol. 2, no. 3, pp. 1–11, 2008.

[15] R. Popescu-Zeletin, S. Steglich, and S. Arbanowski, “Pervasive communication: a human-centered service architecture,” in Proceedings. 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, 2004. FTDCS 2004., 2004, pp. 140–146.

[16] Y. Liu and F. Li, “PCA: A Reference Architecture for Pervasive Computing,” in 2006 First International Symposium on Pervasive Computing and Applications, 2006, pp. 99–103.

[17] G. Roussos and A. Marsh, “A blueprint for pervasive self-care infrastructures,” in Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW’06), 2006, p. 6 pp.-pp.484.

[18] J. Zhou et al., “PSC-RM: Reference Model for Pervasive Service Composition,” in 2009 Fourth International Conference on Frontier of Computer Science and Technology, 2009, pp. 705–709.

[19] A. Fernandez-Montes, J. A. Ortega, J. A. Alvarez, and L. Gonzalez-Abril, “Smart Environment Software Reference Architecture,” in 2009 Fifth International Joint Conference on INC, IMS and IDC, 2009, pp. 397–403.

[20] J. Liao, J. Wang, B. Wu, and W. Wu, “Toward a multiplane framework of NGSON: a required guideline to achieve pervasive services and efficient resource utilization,” IEEE Commun. Mag., vol. 50, no. 1, pp. 90–97, Jan. 2012.

[21] L. C. Bueno, “A Reference Architecture for Component-Based Self-Adaptive Software Systems,” Department of Information and Communication Technologies Faculty of Engineering, ICESI University, Cali, Columbia, 2012.

[22] “Internet of Things Architecture IoT-A Project Deliverable D6.2 – Updated Requirements.” European Lighthouse Integrated Project, Jan-2011.

[23] R. Al Ali, I. Gerostathopoulos, I. Gonzalez-Herrera, A. Juan-Verdejo, M. Kit, and B. Surajbali, “An Architecture-Based Approach for Compute-Intensive Pervasive Systems in Dynamic Environments,” in Proceedings of the 2Nd International Workshop on Hot Topics in Cloud Service Scalability, New York, NY, USA, 2014, p. 3:1–3:6.

[24] I. D. Addo, S. I. Ahamed, S. S. Yau, and A. Buduru, “A Reference Architecture for Improving Security and Privacy in Internet of Things Applications,” in 2014 IEEE International Conference on Mobile Services, 2014, pp. 108–115.

[25] C. A. Machado, E. Silva, T. Batista, J. Leite, and E. Nakagawa, “RA-Ubi: A Reference Architecture for Ubiquitous Computing,” in Software Architecture: 8th European Conference, ECSA 2014, Vienna, Austria, August 25-29, 2014. Proceedings, P. Avgeriou and U. Zdun, Eds. Cham: Springer International Publishing, 2014, pp. 98–105.

[26] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton, “Smart objects as building blocks for the Internet of things,” IEEE Internet Comput., vol. 14, no. 1, pp. 44–51, Jan. 2010.

[27] V. Kostakos, E. O’Neill, and A. Penn, “Designing Urban Pervasive Systems,” Computer, vol. 39, no. 9, pp. 52–59, Sep. 2006.

[28] H. I. Yang and A. Helal, “Safety Enhancing Mechanisms for Pervasive Computing Systems in Intelligent Environments,” in 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom), 2008, pp. 525–530.

[29] O. M. Khaled, H. M. Hosny, and M. Shalan, “On the road to a reference architecture for pervasive computing,” in 2015 International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS), 2015, pp. 98–103.

[30] M. Wollschlaeger, S. Theurich, A. Winter, F. Lubnau, and C. Paulitsch, “A reference architecture for condition monitoring,” in 2015 IEEE World Conference on Factory Communication Systems (WFCS), 2015, pp. 1–8.

[31] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, Discrete-Event System Simulation, 5th ed. Prentice Hall, 2010.

[32] E. Cavalcante, T. Batista, and F. Oquendo, “Supporting Dynamic Software Architectures: From Architectural Description to Implementation,” in 2015 12th Working IEEE/IFIP Conference on Software Architecture, 2015, pp. 31–40.

[33] O. M. Khaled, H. M. Hosny, and M. Shalan, “A Statistical Approach to resolve conflicting requirements in pervasive computing systems,” presented at the The 12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2017), Porto, Purtogal, 2017, p. 12.

[34] S. S. Brink, “Enabling Architecture Validation in the Analysis Phase of Developing Enterprise or Complex Systems using Enterprise Architecture Simulation Environment (EASE),” in MILCOM 2007 - IEEE Military Communications Conference, 2007, pp. 1–8.

[35] R. Roshandel, N. Medvidovic, and L. Golubchik, “A Bayesian Model for Predicting Reliability of Software Systems at the Architectural Level,” in Proceedings of the Quality of Software Architectures 3rd International Conference on Software Architectures, Components, and Applications, Berlin, Heidelberg, 2007, pp. 108–126.

[36] D. Storm, “Of 10 IoT-connected home security systems tested, 100% are full of security FAIL,” COMPUTERWORLD, 11-Feb-2015.

[37] J. De Vries, C. Burki, and B. De Vries, “How to save on software maintenance costs,” Omnext, Nov. 2014.

[38] Y. Liu and I. Traore, “Complexity Measures for Secure Service-Oriented Software Architectures,” in Predictor Models in Software Engineering, 2007. PROMISE’07: ICSE Workshops 2007. International Workshop on, 2007, pp. 11–11.

[39] J. Iqbal, D. S.M.K.Quadri, and T. Rasool, “On Way to Acquiring Reliability Growth in Software Systems,” Int. J. Comput. Appl., vol. 24, no. 7, pp. 33–36, Jun. 2011.

[40] “DEVS-Suite,” Arizona Center for Integrative Modeling& Simulation, 2.0.

[41] H. Pham, System Software Reliability. Springer London, 2006.

AUTHORS

Osama M. Khaled received his Bachelor, Masters and PhD degrees in Computer Science from the American University in Cairo in 1998, 2004, and 2017 respectively. Osama works in the software development and telecommunication industry since 1998 and acts as a lecturer and consultant in software engineering as well. His active research areas are in software engineering, software architecture, software modelling, business analysis, and software programming. Osama is the author/co-author of 12 publications in international journals and conference proceedings.

Hoda M. Hosny is a Professor of Software Engineering and has been teaching at the American University in Cairo since 1985. She started her teaching career in the University of California, Davis (UCD) in 1981. She received her Masters degree in Computer Science from UCD in 1984 and her Ph.D. from Leeds University, UK, in 1991. She served as an IT Specialist, Consultant and Trainer at a number of National and International Institutions and was invited to lecture at 4 other universities in Cairo. She is the author/co-author of more than 50 publications in international journals and conference proceedings.

Mohamed Shalan is an Associate Professor (with tenure) at the Department of Computer Science and Engineering, the American University in Cairo. He received his Ph.D. in computer engineering from the Georgia Institute of Technology (GaTech) in 2003. He received his B.Sc. and M.Sc. in computer and systems engineering from Ain Shams University, Cairo, Egypt in 1993 and 1997. His research interests are in the area of computer engineering, with focus on embedded systems, digital design, energy-efficient computing systems, and electronic design automation. Professor Shalan has over 30 refereed conference and journal papers. Also, he holds 2 US patents.