A SIMULATION APPROACH TO PREDICATE THE RELIABILITY OF A PERVASIVE SOFTWARE SYSTEM

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

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

ABSTRACT

The pervasive computing domain is a very challenging one and requires a robust architectural model to facilitate the production of its systems. In this paper, we explain a case study using a simulation prototype to validate our baseline architecture of a reference architecture for the pervasive computing domain. The simulation prototype was very useful in predicting the reliability and availability of the system using the baseline architecture during runtime.

KEYWORDS

Pervasive Computing, Ubiquitous Computing, Software Engineering, Software Architecture, Software Validation and Verification, Internet of Things, IoT, Discrete Event Simulation

1. INTRODUCTION

Software Architecture is one of the critical tasks in the software development lifecycle. The design decisions that the software architect takes represent the skeleton of the software system. Incorrect decisions may lead to a complete failure of the whole system. Moreover, correcting the wrong designs could be very expensive for both the development team and the customer.

The software architectural model is either a concrete or a reference one. The concrete software architectural model targets a specific problem domain within a specific context. It may not be used in other contexts. On the other hand, the software reference architectural (RA) model targets a specific problem domain which could be used in different contexts with or without modifications in the basic model.

The software development community (SDC) developed some quantitative methods like SAAM, ATAM, and ALMA [1] to evaluate the architectural models. These methods depend mainly on the human factor. There are other quantitative methods which provide more concrete figures about some architectural attributes like cohesion, reusability, and maintainability.

Software architects sometimes need to realize the architecture before implementation since it gives them more confidence about the pursued decisions. Accordingly, the SDC improvised some traceability and experimental methods. These methods try to link the system requirements with

David C. Wyld et al. (Eds) : IPPR, SOENG, DaMi, CSIT, AIS, CSE, CSIP, CCNET - 2018
pp. 59–77, 2018. © CS & IT-CSCP 2018 DOI : 10.5121/csit.2018.80505

the architectural decisions and hence may instantiate concrete architectures, generate use cases, develop prototypes, or even develop complete applications to measure the system architecture coverage against the system requirements.

A simulation prototype is an experimental prototype. 'Simulation' is an artificial activity that tries to imitate an operation in the real world across a period of time [2]. It could be done manually or may be automated depending on the complexity of the simulation operation scenario.

Our research work aimed to generate a software reference architecture for pervasive computing systems from a software engineering perspective (PervCompRA-SE). The reference architecture which was developed contains business and technical sides. The technical part fulfills the needs of the business part by introducing reference models for the smart object, smart environment, and the pervasive system. The baseline reference architecture in pervasive computing explains the basic structure and the behavior of the pervasive system. We used both qualitative and quantitative methods to evaluate the reference architecture. Moreover, we implemented a simulation prototype to realize the baseline architectural model for pervasive systems [3].

The paper is organized as follows: Section 2 covers the work related to the evaluation of pervasive systems. Section 3 explains our approach to build the case study. Section 4 narrates the details of the simulation story. Section 5 explains the conceptual model of the simulation prototype and section 6 gives the specification of the prototype which recognizes the story and the conceptual model. Section 7 lists the assumptions that we used to make the simulation prototype close to reality. Section 8 depicts the different simulation scenarios and Section 9 shows the prediction of the system behavior at runtime with respect to reliability and availability. Finally, section 10 concludes the paper.

2. RELATED WORK

Research efforts to evaluate software architectural models are already being applied. The existing methods are qualitative, quantitative, or experimental.

Angelov et al. [4] reported on an evaluation methodology for a reference architecture that they developed for a B2B e-contracting solution which aims to improve the contracting process between companies. The Researchers adopted the Architecture Trade-off Analysis Method (ATAM) [1] method with some variations. The authors concluded that in order to maximize the benefit from the ATAM process, then they first need to adapt the step of identifying the stakeholders properly based on the maturity level of the reference architecture, whether it is a practical or a visionary reference architecture. Second, they recommended to select a number of scenarios from different contexts, merge them, then prioritize them in a general format.

Graff et al. [5] proposed a variant from the SAAM [1] to evaluate a RA for an embedded software. Their approach is based on real-life projects in one of the leading copier manufacturers. One of the main challenges for their research work was that they needed to evaluate their RA based on concrete scenarios that can hardly attribute their design decisions to their RA which they called RACE. They decided to resolve this issue by asking a simple question while executing each scenario "What is the impact on the reference architecture?"

From another perspective, the evaluation of system architectures is seen as a straightforward task that can be achieved using quantitative figures. Madhusudanan and Prasanna [6] used metrics for pervasive systems that cover key-design aspects in pervasive systems and middleware platforms in specific. For example, the authors evaluate context-awareness for pervasive systems with respect to the number of locations, environment, user activities, time, and physical objects. They evaluate scenarios with respect to location according to the number of used locations in the selected scenario against the total number of locations in the environment. They do the same evaluation for other attributes like no. of devices and activities then build an evaluation graph.

Malik et al. [7] proposed an evaluation framework that differentiates between quantifiable and non-quantifiable characteristics of pervasive systems. Their approach considers different factors from the system, users, context, and environment. Maintainability, security and privacy, infrastructure, and integration Design factors are considered crucial evaluation factors. However, according to their evaluation, a pervasive system will not be successful if it does not meet user needs and considers user-related factors such as demographics, health, and comfort.

Bueno [8] presented a software reference architecture for component-based self-adaptive software systems. She adopted a more concrete approach to evaluate her RA by instantiating a concrete architecture and implementing a software based on it. She ran some test cases with an assumption that the quality of the application at run-time was an indicator of the quality of the architecture which was instantiated from the reference architecture.

Bogado et al. [9] introduced an evaluation framework for software architecture runtime quality attributes. The authors worked on building a discrete-event simulation model that evaluates quality attributes for a software architecture. They built a specification model using Discrete Event System Specification (DEVS) to formalize their model which was then fed into a simulator. The authors claimed that this method was useful in evaluating an architecture in the early stages of the software development lifecycle.

They described a conceptual model for evaluation. This model captures the generic behaviour of the architecture elements. It has a high level element called Architecural Element which is specialized into a Connection Mechanism and Component. The Component is further specialized into Simple Component and Composite Component. The component is the one that carries responsibilities and has a representation at runtime. The Composite Component is composed of Simple Component and Composite Component elements and its behaviour is determined by the simple components. Quality attribute values (QualityAttributeValue) are identified for responsibilities and measurements (Measure) are taken for them.

Almost every project chose a single evaluation approach to work with. We can rarely find a research project that combined different methods to evaluate a concrete architecture or an RA; although the combined view can provide useful insights for the quality of the RA. The RA represents the model with modules that could be evaluated quantifiably while its documentations could be evaluated subjectively. Moreover, very few reference models adopt the simulation prototype as a method to study the impact of the architecture on the behaviour of the system.

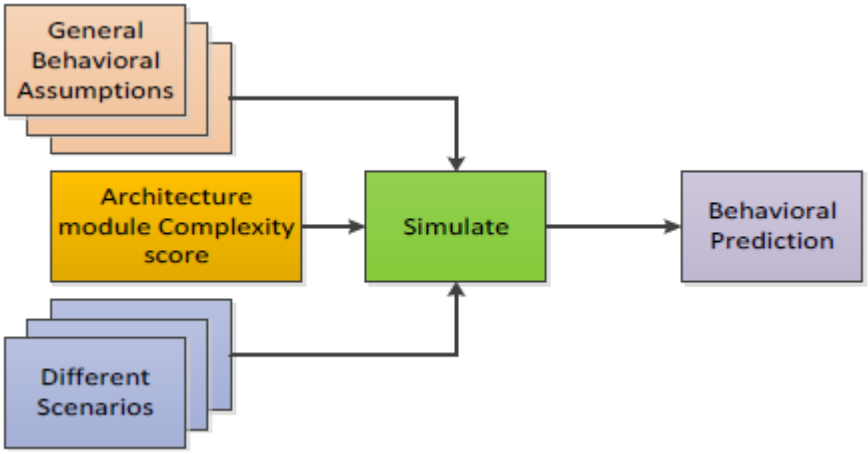

Figure 1. Simulation Experiment (High Level)

3. THE APPROACH

There are several methods to evaluate the structure and the behaviour of a software architecture prior to implementation. These are lead measures which reveal some issues before turning the architecture into implementation. However, some of the issues remain to be discovered after implementation. Accordingly, we need to have a lag measure in order to understand more about the system behaviour at runtime.

Hence, we assessed the reliability and availability of the architecture during runtime by running the simulation experiments. We built a conceptual model and captured state details similar to the ones mentioned in [9]. The results of the simulation were studied to propose enhancements for the baseline architecture, whenever required.

Compared to regular prototype implementations, the simulation prototype does not introduce external variables to the prototype like the hardware, programming language and network. Including these variables would have definitely impacted the final results of the experiment and is time-consuming as well. The simulation model can however clarify the requirements of the user in a virtual space which considers all constraints and quality requirements [10]. The regular prototype implementation may be more suitable for a concrete architecture. In contrast, the simulation approach is better within our scope of research, and at the same time experiments were executed in a more controlled environment. Moreover, pervasive systems are considered quite complex and they cannot follow the traditional development cycle. They need to be tested first using simulation approaches as recommended by Brink [11].

We adopted a discrete-event simulation (DES) approach which can have different states across discrete points in time. Compared to continuous event simulation, a DES simulation approach best fits operations whose states change continuously over time as per Martensson and Jonsson [12].

The real challenge for running simulation experiments to predict the behaviour of a pervasive system, or even a software system in general, is that there is insufficient historical data about similar systems in order to build a robust simulation model [9]. Usually, developers will go for assumptions and opinions from domain experts. Researchers like Roshandel et al. [13] introduced

a software reliability prediction model before implementation based on the reliability of the architecture components. However, their approach requires deep knowledge about the components' design during the design phase. In our approach we ran the experiments based on the technical specifications of the sensors and also on statistics gathered and produced by earlier researchers as well as our scientific calculations for the complexity of the modules as explained in [3]. We executed different scenarios in order to provide a prediction model with a high degree of confidence (Fig. 1).

We developed the simulation prototype using the DEVS-Suite tool [14]. It was fairly easy to customize and even introduce more features to the application by introducing a simple database and building the modules using the Java language.

4. THE SIMULATION EXAMPLE

The simulation scenario that we investigated is a system that studies the quality of the sensors in a bus. There is a bus starting its trip from point A towards point B for a complete 20-hour trip. There is a location and a speed sensor installed on the bus to help the control room detect if the bus has a problem during its trip or not. The system will device its intelligence to make sense of the received data and transform them into meaningful contexts, then interpretations, then decisions, and finally actions. The bus driver will carry out different manoeuvres to stimulate the sensors. For example, he will drive normally then stop suddenly. He can drive normally then slow down suddenly. In other words, he will drive at different speeds with no alarm on when he speeds up or slows down while moving from point A to point B.

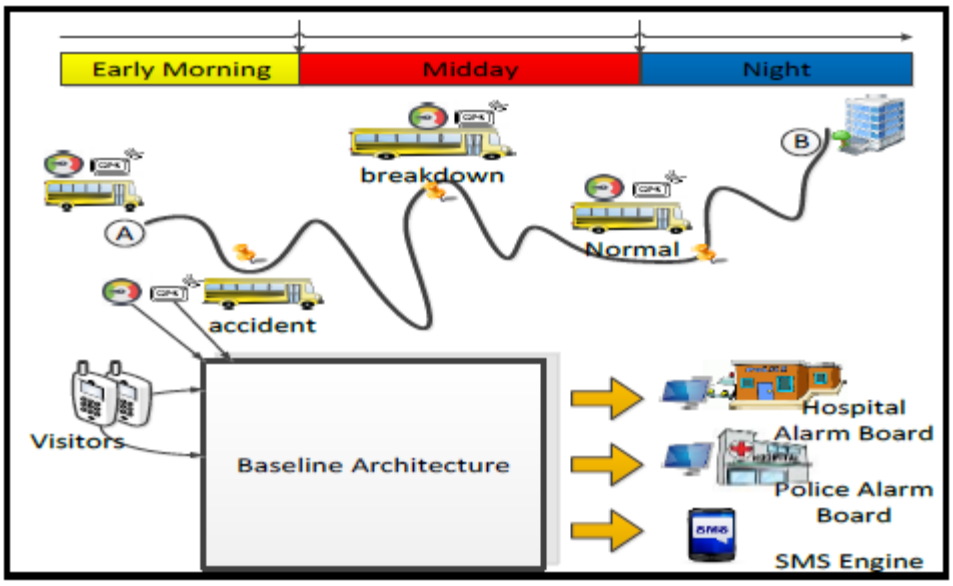

Figure 2. Bus Trip Emergency Study Simulation Example

The data generated from the sensors are classified into specific events:

- **Location Event:** is categorized based on the proximity of the bus from point A and B (At point A, Far away from point B, Midway to point B, Very close from point B, At point B).

- **Speed Event:** is categorized from an accident status point of view (Normal speed, Slowing down, Moving very slowly, Slowed down suddenly, stopped suddenly).

- **Time Event:** is categorized as (early morning, midday, and night).

The different combinations of these events generate a 3-tuple context which derives a specific interpretation. The interpretation drives a decision, which leads into some actions through the system actuators (Digital screens and SMS Gateway).

On the other hand, the system receives visitors who request services. The entities of the system may fail to achieve their duties at some time, but the autonomous error recovery of the system will work on fixing them. Moreover, the system optimization service will monitor the lifetime of the sensors to prolong their lifetime and the rest of the entities to reduce their failure rates. The whole system will be running at different modes in which there are some policies that will be applied (Fig. 2).

5. CONCEPTUAL MODEL

The conceptual model is an important step towards the complete implementation of the simulation prototype. It is derived from the structure of the baseline architecture model as introduced in the original research work [3]. 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. 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. 3).

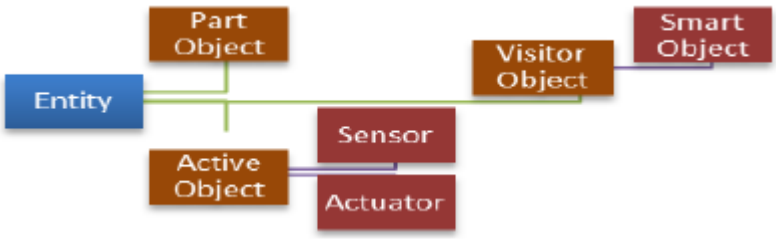

Figure 3. Simulation Conceptual Model

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 at which the entity can accept inputs and generate outputs. All the entities are working on a tick = 10, which is equivalent to one minute, and all the entities have 4 basic phases as shown in Fig. 4:

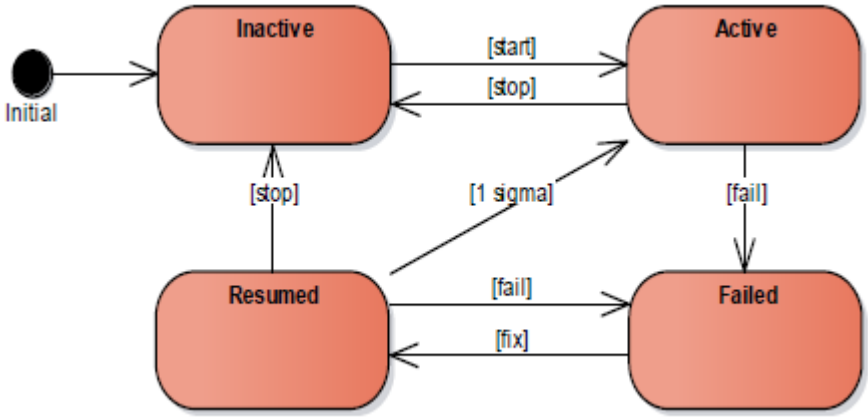

Figure 4. Simulation Module Phases

The whole simulation model can be working in one of the following modes:

1) **Runtime:** It is the normal execution scenario without changes in the settings of the entities.

2) **Assertion:** It is a normal execution scenario but with additional logging activities from these entities. They send data to the Logger to log a specific event.

3) **Security Threat:** In this mode, the whole system is threatened and needs to take some measures 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.

4) **Out of Service:** During this mode 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 work on it.

The state of an entity at any point of time is defined using the 6-tuple (P, AI, AO, L, F, M):

1) **Phase (P):** it is the phase of the entity where P is one of the phases in the set {Active, Inactive, Failed, Resumed}.

2) **Accumulated Inputs (AI):** it is the number of received input requests for all the input ports $AI = \sum_{i=0}^{n} count\ (input_i)$

3) **Accumulated Outputs (AO):** it is the number of submitted outputs for all the output ports $AO = \sum_{i=0}^{n} count\ (output_i)$.

4) **Lifetime (L):** is the lifetime indicator of the entity which takes a value from 0-100. 100 indicates that it is healthy and fully powered, and 0 indicates that it is dead. It is an optional state attribute for part objects

5) **Failures (F):** it is the counter of the failures. It is ceiled by a maximum threshold. The counter will reset to 0 after reaching the threshold. It is an optional state attribute for active objects.

6) **Mode (M):** It is the mode of the system where M is one of the modes in the set {*Runtime, Assertion, Out of Service, Security Threat*}.

6. MODEL SPECIFICATION

The main building modules of the simulation project are derived from the baseline architecture as specified in [3]. In addition, we added more modules to make the prototype more controllable and fulfilling for the simulation example as well.

The prototype starts using the *Simulation Starter* module which is responsible for starting, stopping, changing execution modes, and dumping statistics about the simulation runs. The *Policy Manager* is responsible for applying the system policy on the modules according to the mode of execution. The sensors of the prototype are the *Speed Sensor* and the *Location Sensor*. Both send their data to *Sensor Synthesizers* which receive the data and generate the sensor signals after checking their correctness based on a standard deviation of expected error factors.

The prototype uses two digital boards (*Hospital Alarm Board* and *Police Alarm Board*) and an *SMS Engine*. They are used to send alarms and notifications about accidents that may occur during the simulation. They are the actuators that fulfil actions in order to adapt to the changes in the context of the experiment.

The *Repository Manager* is responsible for storing the synthesized sensors' data in a 3-tuple format including the time, location and the speed. It stores also the profile of the users, which is managed by the *Profile Manager*, and the visits made by the smart objects.

The system makes sense of the 3-tuple data format by transforming it into a 3-tuple context through the *Event Handler*. The system then employs the Interpretation Manager to interpret the 3-tuple context. The *Decision Manager* uses the interpretation to make a decision which triggers actions through the actuators.

The systems simulates the visits of the users through the *Smart Object* which sends visits to the system similar to the attendance models of the employees as it is expected to have more users join the system during the day and the trend decreases slowly where the disjoin trend increases by the end of the day. The *Device Manager* is responsible for registering the joins and leaves of the smart objects. The *Service Manager* receives requests from the visitors and fulfils them through the different modules of the system like the *Repository Manager*, actuators, and the sensors.

Every service has an authorization level based on the smart object type, *visitor* or trusted smart object. If the smart object requests a service that has an authorization level not suitable for its type, then the *Service Manager* rejects the request. The *Risk Handler* is responsible for studying the requests from the smart objects to join the system and puts it on the proper status (*visiting, trusted, prohibited*, or *rejected*). It is also responsible for handling the certificate requests sent from the joining smart objects.

The *Fault Manager* is responsible for handling faults that cause part objects to be out of service. For the sake of consistency and better tracking in the simulation model, the *Fault Handler* is responsible also for failing the modules. It is important to note that the probability of part object failure increases based on its complexity as shown in equation (1) [3]:

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

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. The equation derives the faults from the satisfied requirements, which could be translated as internal part object capabilities and the dependency relationships with other part objects. It is then divided by all the weights of the modules and multiplied by 100 to get a percentage. The weight is rounded off after that.

The *Optimization Manager* is responsible for monitoring the failure rates of the modules and the lifetime status as health performance indicators for the sensors, actuators, and the part objects and takes decisions to recover their performance. The *Resource Manager* receives a request from the *Optimization Manager* to allocate a resource for a nominated part object, or sensor. If the request is to reduce failures, then the *Resource Manager* will select a resource, which could be a hardware or a software, randomly from a set of resources reserved for the part objects only, if not already allocated. The part object receives the resource which gives it a limited protection from failure through a pre-defined period of time, e.g. 100 ticks. Accordingly, if the *Fault Handler* decides to fail a part object that has a resource allocated for it, the part object will ignore this fail message.

If the request is to recover the lifetime of the sensor, then the *Resource Manager* will select a resource randomly from a set of resources reserved for the sensors only, if not already allocated. The battery resources increase the lifetime of the hardware instantly by a specified lifetime value [15].

The data collected by the *Repository Manager* and the *Logger* is analysed by the *Analysis Manager* which shares some of its knowledge with the external entities represented as a module called *Interested Community*.

7. SIMULATION ASSUMPTIONS

There are some important assumptions that had to be designed within the probability model of the simulation prototype. They are derived from real data concerning software and hardware components. These assumptions are embedded in the simulation prototype as settings which could be modified as needed to produce numerous simulation scenarios. In order to make the simulation scenario very close to reality, we made some assumptions derived from actual statistics.

The sensors are assumed to be running on batteries that deplete gradually according to the rate of generated sensor data. They start with an initial capacity of 100%, and decrease gradually by X% (e.g. 0.5%) with every activity. On the other hand, part objects and actuators are running on permanent power, but they may experience random failures every now and then.

We investigated the technical settings of some sensors (battery lifetime, and accuracy for the time pulse signal). The ideal settings for the sensors are as follows:

1) **Speed Sensor**: It is assumed to have a 15 hours battery lifetime in normal conditions [16]. We assume the minimum hours for the battery is 7h12m and maximum is 18h42m [17] and the probability of failure between 10-7 and <= 10-8 per hour [18]. Our settings are derived from two speed sensor products.

2) **Location Sensor:** The same battery lifetime of the speed sensor is assumed here as well for the location sensor [17]. The horizontal position accuracy has a standard deviation of 0.35 meters [19]. The assumptions are derived from one product.

We assume that the probability of failure for the SMS Engine is 0.05 as per the research on the reliability of short messaging [20]. The failure rate of the SMS Engine as an active object is independent from the part objects of the system.

We assume that the failure rate for the digital screens installed in the Police department and the Hospital is 0.03 due to product defects as stated by Shaw [21]. Given other external factors like scheduled maintenance, power supply cut-off, and software failure, then we can safely increase the failure probability to 0.05. The digital screen failure rate is quite independent from the failures of the part objects.

Part objects are assumed to be running on different servers from the same manufacturer and the same manufacturing year. We assume that the best probability of total failure for the part object at any minute is 0.05 based on estimates from [22] [23]. On the other hand, another interesting research, by YAN [24], shows that the reliability of a pervasive system can be less than 0.5. So, we assign 0.5 as the worst probability. The average in this case will be 0.275. The values for the control variable (Part Object Failure Optimization Threshold) as shown in Table 1 are based on experience with IT support units in the Telecom industry.

Pervasive systems, or IoT systems, are highly vulnerable to security threats [25]. 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 [26]. Moreover, administrators dump logs from the system for monitoring purposes all the time. Accordingly, we assume that the system will be running in normal mode most of the time (64-70%) and that there is 30-36% probability that it will be running in one of the other abnormal modes (Assertion, Out of Service, or Security Threat).

The *Optimization Manager* checks the status of the battery if it reaches 40% of its capacity [27] on average. We assume that the mean time of repair for the part object is shorter than the mean time between failures [28]. We assume that the more complex is the part object, the higher the probability of failure, and the less complex is the part object the faster we can get a repair [29] [30].

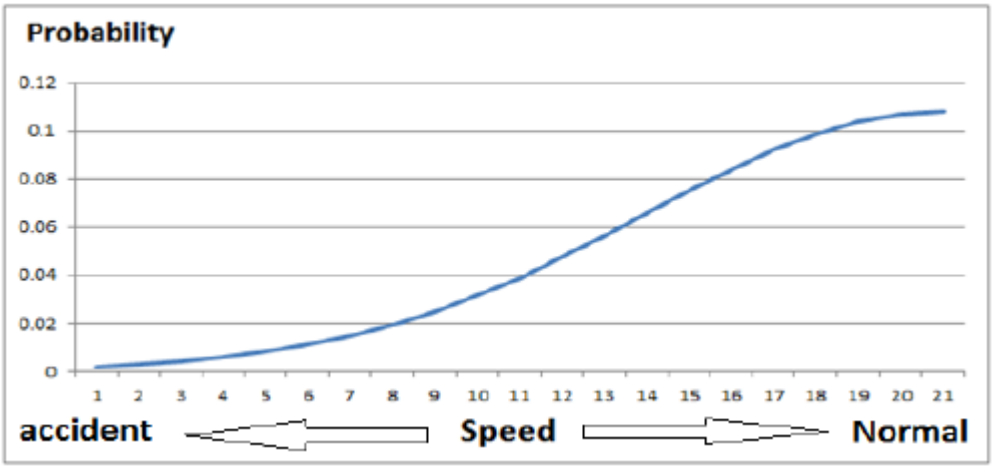

Figure 5. Speed normal probability function

We also assume that the probability of having a normal trip is normally distributed around normal driving and that the accidents are rare (bell curve shape) with very low probability as reported by some studies around accidents in the USA [31] [32]. We executed different runs to generate values by the Gaussian function, and optimized the standard deviation σ in order to get a bell curve that fits the probability distribution model for normal driving and accidents. We used average = 21 and σ = 7. Anything greater than 3σ and less than -3σ is set to 0, otherwise values between average and 3σ are mirrored to be in the range of (0, average) as shown in equation 2.

$$RN = Gaussian\ Random\ Number * \sigma + average$$
$$RN = \begin{cases} \mathbf{0}, & RN > 3\sigma \ or \ RN < -3\sigma \\ \boldsymbol{RN - 2 * (RN - average)}, & RN > average \quad and \ R < 3\sigma \\ \boldsymbol{RN} \end{cases} \tag{2}$$

The function generates a semi-bell probability shape as shown in Fig. 5 where 0 is an indication for an accident and 21 is an indication for a normal driving speed.

We assume that the visits of the smart objects are tightly coupled with the visitors' behaviour to the attendance of employees in a workplace. It is assumed that the human visitors' trend has peak visits during the early morning and decreases through the day. Accordingly, the smart objet disjoins the system across the day but there is a peak at the end of the day when visitors start to leave the school. During this time, visitors can request services in a normal distribution where the most aggressive period is at midday (Fig. 6).

The simulation model generates new join requests from smart objects as shown in equation/algorithm (3) [3]. We assume an average of 6 visits at a time and σ = 3. It generates a large number of visits, not exceeding 7000, if the simulation model will execute for 1500 ticks. The same algorithm will generate disjoin requests but at very small rates in the early day time and increasing by the end of the day.

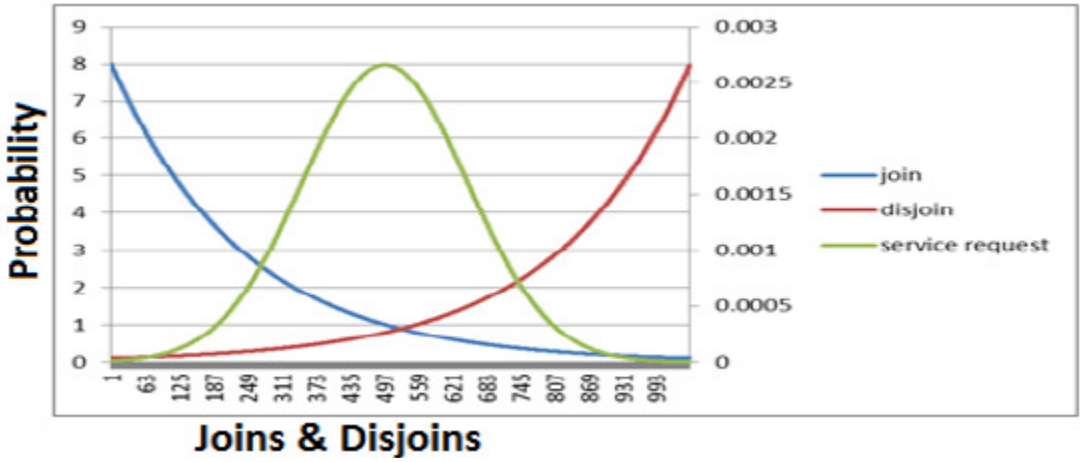

Figure 6. Smart Objects join/disjoin behavior during the simulation

Loop from $tick = 1$ to N

$$V = Gaussian\ Random\ Number * \sigma + average$$

$$V = \begin{cases} \mathbf{0}, & V > 3\sigma \ or \ V < -3\sigma \\ \boldsymbol{V - 2 * (V - average)}, & V > average \ and \ V < 3\sigma \\ \boldsymbol{V} \end{cases} \quad (3)$$

$$join\ requests = round\left(V * \left(1 - \frac{tick}{N}\right)\right)$$

$$disjoin\ requests = round\left(V * \left(\frac{tick}{N}\right)\right)$$

End Loop

The assumptions that we presented are accurate to our best knowledge and based on credible references. We evaluated the best and worst values in order to use them in our simulation experiments. They are all derived from the same sources as summarized in Table 1, given that the average may not represent a calculation from the best and worst values.

8. EXPERIMENTATION SCENARIOS

In order to provide an acceptable prediction model for the reliability and availability of the PervCompRA-SE, we implemented different simulation scenarios. We calculated the Mean Time between Failures (MTBF) and Mean Time to Repair (MTTR) in order to calculate the reliability and the availability scores of the simulation scenario.

There is the perfect scenario which assumes that the bus starts from point A to point B and that the system completes processing all the sensor data on time *Tp*. There are no failures in the system and the smart objects' requests are all satisfied, and batteries do not deplete. The perfect scenario will be used for benchmarking purpose. This scenario does not require a *Fault Handler* nor an *Optimization Manager*. It will be executing always in the *Runtime mode*.

There is the **Normal Hybrid Modes** scenario which assumes the values in Table 1. There will be faults and repairs in that scenario. The scenario introduces more disturbances to the normal flow of the execution cycle by introducing changes in its execution modes. It is expected that it will take a longer time than scenario 2. There will be 3 runs for each category of values (Best, Average, and Worst). It should finish at time ($Tp + \Delta Tnb + \Delta Tnbh$), ($Tp + \Delta Tna + \Delta Tnah$), and ($Tp + \Delta Tnw + \Delta Tnwh$), respectively. We will assume a fixed number of resources (Rn = 12) across all the runs.

Table 1. Assumed values of Control Variables

Value Boundary / Control Variable	Average(A)	Best (B)	Worse(W)
Speed Sensor signal failure rate	9.167E-09	1.67E-10	1.67E-09
Speed Sensor battery lifetime degradation /minute	0.001	0.0009	0.002
Location Sensor signal accuracy (per meter)	2.25	2	2.5
Location Sensor battery lifetime degradation (per minute)	0.003	0.0009	0.002
Battery Recharge Threshold (%)	0.4	0.5	0.2
Part Object Failure Optimization Threshold	2	1	3
SMS Engine Failure rate	0.05	0.1	0.16
SMS Engine Repair rate	0.95	0.9	0.84
Hospital Alarm Board Failure rate	0.05	0.025	0.075
Hospital Alarm Board Repair rate	0.95	0.975	0.925
Police Alarm Board Failure rate	0.05	0.025	0.075
Police Alarm Board Repair rate	0.95	0.975	0.925
Runtime Mode Rate	0.67	0.7	0.64
Part Object Failure Rate	0.275	0.05	0.5
Part Object Repair Rate	0.725	0.95	0.5
Accident Rate	0.004	0	0.03

There is the **Normal No-Optimization** scenario which aims to predict the behavior of the technical model without the optimization mechanisms (*Optimization Manager, Resource Manager*). There will be 3 runs for each category of values (Best, Average, and Worse). It is expected that the processing time for this scenario will take additional time ($\Delta Tnoop$) for every group of runs than scenario 3. It is expected also that the MTBF will increase more than what is recorded in scenario 3.

There is the **Normal resource-optimized** scenario which aims to predict the impact of the number of resources on the system reliability. The scenario will show the impact of the *Optimization Manager* and the *Resource Manager* on the number of faults that the system may encounter. It is also expected to see some decrease, ($-\Delta T_r = 4$), ($-\Delta T_r = 8$), and ($-\Delta T_r = 12$), in the

processing time relevant to the number of resources, than scenario 3 and increased time between failures ($MTBF + \Delta T_f$). There will be 3 runs for each category of resources using the Average control variables. It is important to note that the last scenario variation is the same as scenario 3 with Average control variables.

Finally, there is the **Extreme** scenario which aims to predict the behavior of the technical model under extreme conditions. We will use the extreme values, which exceed the boundaries of the best and worst, to run two categories of runs (Extreme Best and Extreme Worst). A fixed number of resources as in scenario 3 ($R_{ex} = 12$) is assumed here. The best and worst values in Table 1 are considered as the standard lower and upper boundaries. We built a simple capability model to stretch the best and worst boundaries as follows [33]:

1. We calculate the average from the lower and upper bounds.
2. Calculate the standard deviation (σ).
3. We calculate the minimum value, whether it is best or worst, as (min=average-3* σ).
4. We calculate the maximum value, whether it is best or worst, as (max=average+3* σ).
5. If the value exceeds the logical or physical limits, then it is set to the maximum possible value.

9. ANALYSING THE RESULTS

A reliable system is a system that can perform its assigned functionality with a high probability during a specified period of time and within specific design constraints [13]. A reliability measurement is a function in MTBF and gives a score between 0 and 1 [28] as shown in equation (4) [3]. On the other hand, software availability is the probability of the uptime of the system. It is a function of MTBF and MTTR [28] as shown in equation (5) [3]. For example, if we measure the availability of a website during a year and it is 0.99, then it means that the system downtime was (3.65 days) calculated as ((1-availability) x 365). MTBF measures the average time between successive failures without considering the time taken to repair the system in order to reflect its ability to fulfil its duties. If a system's reliability is 0.99, it means that the system is expected to run successfully from time 0 to time *t* with probability 99%.

$$Reliability = \frac{MTBF}{MTBF + 1} \qquad (4)$$

$$Availability = \frac{MTBF}{MTBF + MTTR} \qquad (5)$$

The experiments show some facts about the technical baseline architecture:

1) The experiments predict the reliability of the architecture in the worst case as 96.86% and the availability as 90.89%.

2) In the extreme worst cases both reliability and availability measurements decrease noticeably as reliability becomes 92.62% and availability deteriorates to 31.83%.

3) On average the system availability is 95.77% and reliability is 98.08% if we exclude the Perfect and extreme cases.

4) In the best cases, the system availability is 99.79% and reliability is 99.9%.

The results show that that there are variations in processing time among all the scenarios. We predict an average of 2% additional processing time as overhead from the last sensor input calculated against the perfect scenario (Fig. 7). The results show that the resource optimization technique that we adopted is working reasonably. The experiments show an average of 3.09% immunity from failures across all the scenarios. As the resources allocated increase, the immunity of failure provided to the system increases as well. In general, the scenarios show that the processing time increases as the working conditions get worse.

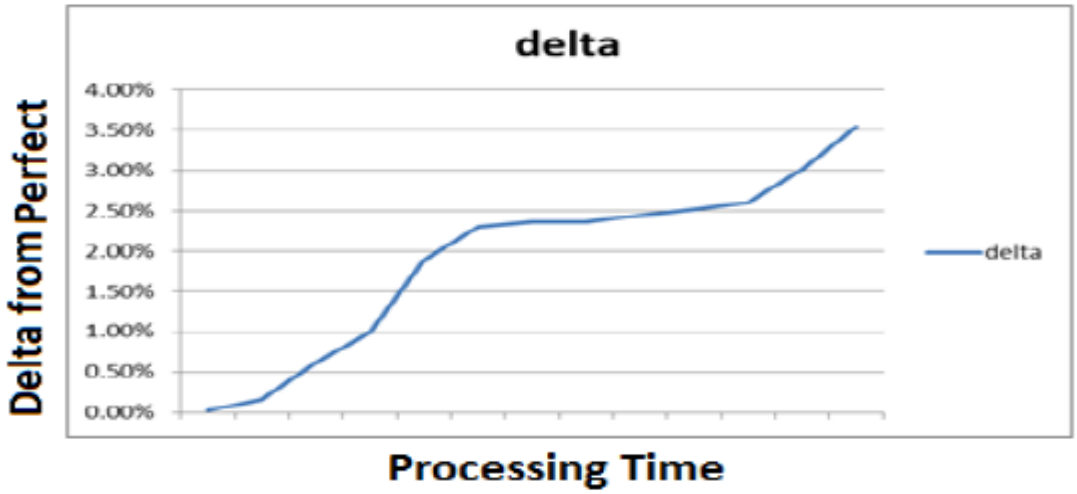

Figure 7. The processing time overhead for the simulation scenarios compared to the perfect scenario

10. CONCLUSION

The simulation case study presented in this paper was used to evaluate a baseline architecture as part of an overall evaluation of a reference architecture. There was a hypothetical example (bus story) to track the behaviour of the architecture. We assigned roles and responsibilities for the modules of the baseline architecture through a specification exercise in addition to adding other modules to introduce more controls to the simulation prototype. The prototype adopted some probability models through assumptions inferred from collected statistics about software and hardware components. The simulation prototype is then executed in different scenarios. The results were then analysed and the reliability and availability of a system adopting this baseline architecture are shown to be over 90%.

There are modules that we did not expose to failures and repairs during our simulation exercise (Repository Manager, Logger, Fault Handler, and Synthesizers). These are common modules for all the other modules in the system. They do not need a simulation exercise to understand that a single failure in the Repository Manager will hinder the overall stability of the whole system. It is very clear also that the Logger can impact the overall performance if it is not responsive. Moreover, the Fault Handler is designed to respond to failures, and it is essential to make it more reliable and available than other modules. Finally, the Synthesizer is either a part of the sensors

and actuators hardware or it is at a low-level software layer that the sensors and actuators must interact with. If this layer fails, then the data may be corrupted.

Although the analysis shows positive results about the reliability and the availability of the architectural model, the prediction is an initial estimation which will definitely change in a real environment. This 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 probability of the system failure.

It is quite beneficial for the architects to have an accessible simulation package for the reference architecture model containing configurable settings for all the control variables. The simulation package should help the architect represent the different use cases as he/she builds a real-life scenario using the PervCompRA-SE.

ACKNOWLEDGEMENTS

A special gratitude goes to Professor Veronica Bogado in the *Departamento Ingeniería en Sistemas de Información*, UTN FRVM for providing the necessary material that helped with this simulation project.

REFERENCES

[1] M. A. Babar and I. Gorton, "Comparison of scenario-based software architecture evaluation methods," in 11th Asia-Pacific Software Engineering Conference, 2004, pp. 600–607.

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

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

[4] 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.

[5] B. Graaf, H. van Dijk, and A. van Deursen, "Evaluating an Embedded Software Reference Architecture — Industrial Experience Report —," in Ninth European Conference on Software Maintenance and Reengineering, 2005, pp. 354–363.

[6] J. Madhusudanan and V. Prasanna Venkatesan, "Metrics for Evaluating Pervasive Middleware," Int. J. Intell. Syst. Appl. IJISA, vol. 6, no. 1, pp. 58–63, Dec. 2013.

[7] Y. Malik, M. Soliman, and B. Abdualrazak, "Towards an Evaluation Framework for Pervasive Computing System," in International Conference on Modeling, Simulation and Visualization Methods (MSV), Las Vegas, USA, 2011, p. 8.

[8] 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.

[9] V. Bogado, S. Gonnet, and H. Leone, "A Discrete Event Simulation Model for the Analysis of Software Quality Attributes," CLEI Electron. J., vol. 14, no. 3, Dec. 2011.

[10] S. P. Miller, "Proving the Shalls: Requirements, Proofs, and Model-Based Development," in 14th IEEE International Requirements Engineering Conference (RE'06), 2006, pp. 266–266.

[11] 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.

[12] F. Mårtensson and P. Jönsson, "Software Architecture Simulation – a Continuous Simulation Approach," Master's thesis, Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Sweden, 2002.

[13] 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.

[14] "DEVS-Suite," Arizona Center for Integrative Modeling & Simulation, 2.0.

[15] B. Lawson, "A Software Configurable Battery," presented at the EVS26 International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium (EVS26), Los Angeles, California, 2012.

[16] "Speed Sensor Edge 520." [Online]. Available: https://www8.garmin.com/manuals/webhelp/edge520/EN-US/GUID-F50056D5-6DC6-43D2-81A6-61095620E142.html. [Accessed: 21-Apr-2017].

[17] K. Byrne, "Best phone battery life 2016: Top smartphones tested," Expertreviews, 02-Nov-2016.

[18] "SIL3 Speed Sensors." [Online]. Available: http://www.jaquet.com/site/assets/files/1218/flyer_sil3_a4_en.pdf. [Accessed: 21-Apr-2017].

[19] "NEO-6 u-blox 6 GPS Modules." [Online]. Available: https://www.u-blox.com/sites/default/files/NEO-M8_DataSheet_(UBX-13003366).pdf.

[20] X. Meng, P. Zerfos, V. Samanta, S. H. Y. Wong, and S. Lu, "Analysis of the Reliability of a Nationwide Short Message Service," in IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications, 2007, pp. 1811–1819.

[21] M. Shaw, "Reducing LCD TV Warranty Claims Through an Organized and Aggressive Approach to Sub-Assembly and Full LCD TV Assembly Accelerated Stress Testing," presented at the International Applied Reliability Symposium, EUROPE, Milan, Italy, 2010.

[22] "Frequency of server failure based on the age of the server (per year)." [Online]. Available: https://www.statista.com/statistics/430769/annual-failure-rates-of-servers/. [Accessed: 21-Apr-2017].

[23] O. Bäckström, J.-E. Holmberg, M. Jockenhövel-Barttfeld, M. Porthin, A. Taurines, and T. Tyrväinen, "Software reliability analysis for PSA: failure mode and data analysis," Nordic Nuclear Safety Research (NKS), NKS-341, ISBN 978-87-7893-423-9, Jul. 2015.

[24] L. YAN, "Applying Model Checking to Pervasive Computing Systems," Ph.D. Dissertation, Department of Computer Science, School of Computing. National University of Singapore, Singapore, 2014.

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

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

[27] S. Jary, "How to properly charge a phone's battery: stop charging from zero to 100% and other tips," TechAdvisor, 26-Jul-2016.

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

[29] 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.

[30] 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.

[31] "Traffic Safety facts Research Note," U.S. Department of Transportation, National Highway Traffic Safety Administration, Aug. 2016.

[32] D. Toups, "How many times will you crash your car?," Forbes, 27-Jul-2011.

[33] D. S. Moore, G. P. McCabe, and B. A. Craig, Introduction to the practice of statistics : extended version, 6th ed. New York: W.H. Freeman, 2009.

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.