International Journal of Computer Applications (0975 - 8887)
Volume 180 - No.20, February 2018

A Proactive Self-Adaptive Framework for Context-Aware Mobile Applications

Md. Shafiuzzaman
Institute of Information Technology, University of Dhaka

Amit Seal Ami
Institute of Information Technology, University of Dhaka

Rayhanur Rahman
Institute of Information Technology, University of Dhaka

ABSTRACT

Today's mobile applications are continuously renovating user experiences by processing underlying contextual information of the hand-held devices. The mobility provided by these devices is encouraging users to install and use applications on the move. As a result, underlying context of these devices keep changing constantly. As the changes in context are beyond control of the application, a context-aware application needs to be self-adaptive in the sense that it can update its services dynamically in runtime to meet overall user requirements. With the remarkable technological advances of mobile devices, collecting contextual information is no longer an issue. However, adapting the contextual changes are still an open problem as most of the existing self-adaptive frameworks are not proactive in nature and the existing proactive frameworks suffer for their computational complexities as they rely on architectural reconfiguration. As a result, none of those have become effective enough for mobile applications. In order to tackle the aforementioned challenges, this paper proposes a light-weight proactive solution that suggests to offer one or more alternative features of each functionality so that the affected feature can be disabled or reconfigured with another mutually exclusive feature depending on the contextual requirements. Several experiments were conducted to test the achieved context-awareness as well as the imposed overhead. The tests were conducted over battery consumption as it is the most common context for most of the mobile applications. The proposed framework extended the battery life 30.46% than the non-adaptive application. The activity of stopping and activating features had no significant impact on the battery life as only 1.5% battery was consumed when we deactivated 7 features at once. The time required to stop and active features increased linearly with the number of features though it was in the range of milliseconds. It took 0.6 seconds to resume the application after deactivating 7 features simultaneously.

Keywords

Context awareness, self-adaptive system, mobile application, feature-orientation, model-driven development

1. INTRODUCTION

Mobile software systems are characterized by a high degree of unpredictability and dynamism as those can be installed and used anywhere and anytime. At the same time, such systems require strict reliability requirements in the sense they never go out of service. But the traditional mobile applications fail to provide uninterrupted service during frequent contextual changes as they are not non-adaptive in nature. A self-adaptive system is characterized with the capability of re-configuring a software system at runtime to meet expected Quality of Service (QoS) [9]. In practice, engineers try to ensure this capability by explicitly programming alternative behaviors and implementing heavy exception handling mechanism [7]. This is quite unrealistic to ensure exception handling for each combination of different contexts. In addition, using this approach, the architecture and the code become unmanageable as the alternative behaviors cannot be kept separated from each other [7].

In literature, quite a few approaches [6, 8, 9, 11] can be found that target the challenges of self-adaptive systems. Initial approaches were mostly confined into ensuring adaptivity by updating configuration parameters dynamically at runtime [11, 12]. These parameterized adaptation became popular for its simplicity but failed to support large-scale adaptation [9]. Other approaches can be divided into two broader domains - (i) Architecture-based adaptation [6,8] and (ii) Feature-oriented adaptation [9,17]. Architecture-based adaptation depends on adding or replacing architectural components at runtime. On the other hand, feature-oriented adaptation does not require knowledge about internal structure of the software system instead it pulls out the affected feature when it goes out of service and restore the system with an alternative feature simultaneously. Both of the approaches failed to fulfill required stuffs of self-adaptation for mobile applications. Searching for an optimal configuration at the architectural-level is computationally very expensive for mobile applications. In fact, many architecture-based optimization algorithms are shown to be NP-hard [21, 22]. On the other hand, feature-oriented adaptation suffers from proactive decision making as it lacks of knowledge about the execution flow. A mobile application becomes unusable if context changes cannot be adapted before reaching failure state [8].

The proposed approach brings about an innovation for solving the aforementioned challenges. It extracts analytical model [24] from software requirements models to estimate the probable execution flow and assess the context requirements of running application against the ideal requirements mentioned in requirements model. According to the assessment result, it updates the feature set available for the users. The novelty of this approach is to use of requirements model in feature-oriented adaptation to provide adaptivity.

Extraction of analytical model from traditional software requirements models is formalized in our another contribution [1] where we propose a method of modeling and representing context-aware self-adaptive software systems by extending the 'Model-Driven Development' framework [5]. This paper extends that approach to model the overall system as a workflow of some abstract functionalities while each of those can be implemented in one or more alternative ways. Then, all these alternatives merged into a single Markov Decision Process (MDP) [15] with a finite state automaton where each state of the automaton represents an alternative feature of an abstract functionality and the paths represent possible execution flows denoting the probabilities of execution of those states. At runtime, the system is treated as a variable so that features can be enabled and disabled dynamically on availability of contextual factors. To automate this process, an interpreter navigates the automaton developed at design phase state by state and offers the best possible set of features to the users depending on the availability of the contextual requirements.
The proposed framework is evaluated by identifying the overall usability increased by the framework. Besides, we also test the runtime overhead imposed by the framework. This is done by varying number of abstract funcionalities and number of alternative features suggested by [23].
The remainder of this paper is organised as follows: Section 2 describes the existing works of this field. Section 3 presents the framework in detail. Section 4 validates the performance of the framework. Finally, section 5 concludes the paper with future work directions.

2. RELATED WORK

Over the past decade, researchers have contributed with different methodologies, techniques, strategies and frameworks to engineer the adaptation logic. In this section, we provide most notable and relevant approaches to our work.
Bennani et. al [11] propose a parameterized model that reconfigures system's behavior at runtime by measuring system characteristics (e.g., workload). Ryutov et al. [12] also use parameterization to develop adaptive access control and trust negotiation framework. Though parameterized models were welcomed for their simplicity, they lack to support large-scale adaptation [9].
There are several noteworthy approaches that stand on structural changes, such as adding, removing, and replacing software components [6], changing system's architectural style [13] and rebinding component's interfaces [14]. This paradigm of adaptation strategy is commonly referred as architecture-based adaptation. The most recent and notable approach of this domain is proposed in [8]. It proposes a three-layer (Goal management, change management and component control) reference model to manage structural changes. Usually, these approaches last from design time to runtime. At design-time, architectural representation of the system are used to create analytical models (i.e., Queuing Network models [16] and Markov models [15]). Later, the analytical model is used to verify the adaptation objectives by examining the collected data. Though these approaches have achieved noteworthy success in many domains, there are some frailties in this framework. For instances, internal structure of a software component may not be completely discovered at design time and searching for an optimal configuration at architectural level is computationally very expensive [9] and are not be suitable for mobile devices [10].
Ghezzi et. al [18] propose an interesting adaptation strategy that takes advantage of requirements models to ensure QoS for non-functional requirements (reliability and performance). The method continuously checks if there is any divergence from requirements models, if any then it alerts for adaptation. This approach limits its contribution within adaptation alert, though it sets a new era of research on adaptation frameworks.
Enabling and disabling features at runtime to reconfigure the system is also a popular paradigm of self-adaptive software development research. This paradigm is commonly called as 'Feature-Oriented Adaptation' [17]. A recent solution of this paradigm [9] proposes a learning-based framework for run-time management of feature selection and deselection. This paradigm is computationally less expensive than architecture-based approach and also popular for simplicity, though it has less accuracy on taking adaptation decision as it is not aware of any architectural parameters [7] as well as it can not determine a potential failure proactively [18] .

3. APPROACH OVERVIEW

A software system is built on several user requirements and an user requirement can be interpreted as several sequential user activities. Each of those activities can have several different implementations. Traditionally, while designing an application, software engineers choose an implementation approach by making a trade-off between cost and user expectations. Therefore, traditional software systems provide a static set of features. In contrast, this framework proposes to implement a functionality in several alternative ways so that a feature can be taken on or off in running application depending on the condition of contextual requirements. As a result, users may not deal with a similar pool of features each time while launching the application but they will never face any service interruption until complete shutdown of the device. Users must be aware of this dynamic behavior of the application through *Service Level Agreement (SLA)*. To implement this dynamism, we propose to make use of *Model-Driven Development* framework [5]. In this development framework, software engineers model a feature targeting the requirements and map it to a part of the software system. We make use of those requirements models in our framework by injecting the contextual factors into them and also keeping them alive at runtime to assess any QoS violation. For this purpose, the requirements models are transformed to analytical models using the transformation approach described in [24] which are used to verify the running application against the requirements models at runtime. Using this verification result, the application then adapts itself by re-configuring the available feature sets.
Injecting self-adaptivity with applications requires efforts in both design and implementation phase of the development life-cycle. The design phase modeling is based on generating feature selection and modeling strategy while runtime adaptation relies on context collection, analysis and execution of dynamic feature selection. Following sub-sections describe these two phases of the framework in detail.

3.1 Design time

To employ self-adaptability in a running application software engineers are required to perform some additional task while designing the application. This starts with modeling the user requirements with some alternative features. It also modifies the conventional requirements models to make space for integrating contextual factors within the model. Finally, it outputs an analytical model which will be used by the running system to make decision about adaptation. The design phase of the framework consists of following steps:

(1) Feature Modeling
(2) Feature Priority-Adaptability Mapping

(3) Context Injection

(4) Analytical Modeling

3.1.1 Feature Modeling. The input of this feature modeling process is a set of abstract functionalities

$$A = \{A_1, A_2, ..., A_n\}$$

which are identified as a part of requirements engineering process. These abstract functionalities are processed through following work-flows to model more than one concrete implementations for each of them.

3.1.1.1 Identification of the Contextual Factors. At design time, contexts can be captured by analyzing each of the abstract functionalities separately. For this reason, engineers need to use their domain knowledge or their past experiences on observed behavior of similar functionalities. The set of contextual factors can be formally notated as C. We have partitioned this set into two disjoint sets:

—Static context, C_S

—Dynamic context, C_D

C_S captures the set of stable contexts, which will not change later at runtime. C_S may include the known physical constraints that regulate certain environmental phenomena. C_S must be considered while designing the system as some architectural decision may depend on them.
C_D instead captures the contextual factors that are likely to change over time. C_D must be monitored at runtime to check if they are available with their minimum requirements.
We exclude C_S from our framework as they are not likely to change at running application whereas C_D should be considered for their dynamic characteristics at runtime. Therefore, we inject C_D into the requirements models. As only the C_D are considered in our framework, we will describe C_D as C throughout this paper. So, We may define C_D as

$$C = \{C_1, C_2, ..., C_n\}$$

.
Here, each $C_i \in C, 1 \leq i \leq n$ signifies an unique context which characterises the runtime behavior of each abstract functionality. The relation between the contexts and the abstract functionalities can be formally defined as equation 1.

$$A_i \in A, \qquad \mu : P(C) \rightarrow A \tag{1}$$

where $1 \leq i \leq n$ and P(C) denotes the power set of contextual factors and μ represents the combine effect of P(C) on each abstract functionality. Here, the power set of contextual factors is used as every context may not affect each of the functionalities.

3.1.1.2 Determination of the Levels of Contextual Factors. Level is a scale of amount that is usually used to define the categories of any measurement. Engineers need to determine the levels of each of the contextual factors as the alternative features will be modeled targeting each of those levels later on. Here, we propose to use relative measurement instead of actual values as the real context values are difficult to measure and also device dependent. On the other hand, the relative values can be determined easily by comparing the minimum requirements of each of the alternative features. Furthermore, these relative levels can be updated easily later on by obtaining real data from the running system.

For this reason, the proposed framework defines the levels of the contextual factors as the subset of *Natural Numbers, N*. This is formally defined as 2:

$$l_i \in N, \qquad 1 \leq i \leq n \tag{2}$$

So, a level of context lies between $0 \leq l_i \leq n$. Level of a context becomes '0' when a context is insensitive to a feature.

Listing 1: Implementations of two alternative features

```
@Functionality (name = "A1" )

@Feature (name = "F1" )
@Level (factors = {"C1", "C2", "C3"}, values
    = {0,1,2})
implemetation_F1 (input) {
   // Implementation
}

@Feature (name = "F2" )
@Level (factors = {"C1", "C2", "C3"}, values
    = {1,2,3})
implemetation_F2 (input) {
   //Implementation
}
```

3.1.1.3 Modeling Alternative Features. In this step, engineers need to design one or more alternative features for each of the abstract functionalities targeting the different levels of the contextual factors. Therefore, engineers need to model a feature for each μ defined in equation 1. The relation between contextual factors, abstract functionalities and their alternative features can be formalized using equation 3. Here, F denotes the set of features of an abstract functionality,

$$F = \{F_1, ..., F_n\}$$

such that each of the members of this set implements the abstract functionality targeting each μ.

$$F, C \models A_i \in A \tag{3}$$

That is, a F ensures satisfaction of an abstract functionality $A_i \in A$ in the context of C.
To visualise the abstract view of a feature, we prepare a template for expressing the alternative features. For this purpose, we use the adhoc annotation *@Functionality* to express the functionality, *@Feature* to define each of the features of that functionality and *@Level* to express the levels of the contexts. An example template of expressing the feature modeling is illustrated in Listing 1

3.1.2 Feature Priority-Adaptability Mapping. Frequent updates of available feature sets at runtime may create extra overhead on context information acquisition and processing for decision making. To tackle this issue, the framework presents a trade-off that updates the feature sets considering their priority and importance. For this reason, it proposes to make use of *Feature Prioritization Policy* [25]. Prioritization relies on giving more weight to most important features. Following requirements engineering approach [25], we have defined a list of priority types, such as:

—Critical Features, F_C

—Important Features, F_I

—Useful Features, F_U

Additionally, we propose to annotate all features in F either required (F_R) or optional (F_O) feature group. The priority type and the feature group can be formally defined using following enumeration:
Priority List = <F_C, F_I, F_U>
Feature Group = <F_R, F_O>
F_C always lies in F_R feature group as service of those features can only be degraded in critical situation but shouldn't be unavailable until a complete shutdown of the device. On the other hand, F_I is under control of designer's justification as it can be programmed as running with degraded service as well as making features unavailable as a last resort to adapt to context changes. For this reason F_I can stay in both of the feature groups F_R and F_O. F_O features are not subjected to feature degradation, those will be unavailable in critical situations. As F_U is nice to have features, they are in F_O feature group. They are not subjected to feature degradation instead either the context allows their activation or make them unavailable. Formally defined, priority-adaptability mapping takes priority type and feature group of each of the features and maps them to the possible adaptability that needs to perform. Table 1 identifies the mapping we are suggesting where the critical feature can only be adapted by degrading their service. Important 'required' features should be degraded at first but they can be made unavailable in worst cases. Important 'optional' features and useful features should not be programmed to provide degraded services instead they should be taken off if needed.

3.1.3 Context Injection. In this phase, requirements models are updated to fit the features modeled in *Feature Modeling* phase. This phase uses the *Feature Priority-Adaptability Mapping* as an input so that the requirements models can be updated using this mapping. Now-a-days, varieties of requirements models used in requirements engineering phase depending on the nature of the applications. Here, we examined our framework using UML activity diagrams as it is the most common requirement model used in different software projects, though other models can fit here easily.
As UML activity diagrams organize the functionalities provided by an application in workflows, the context can be easily inserted into the decision states of the activity diagrams. The proposed framework insist software engineers to follow following steps to inject contexts in requirements models:

(1) At first, engineers design separate activity diagrams for each of the features
(2) Then, each of the alternative features of an abstract functionality is merged into a single activity diagram
(3) In overall activity diagram, different functionalities are enumerated using 1, 2, ... while each of the alternative features are annoted using A, B, C....

A prototype of this extended activity diagram is illustrated in Fig. 1. The illustrated activity diagram models three abstract functionalities (1, 2 and 3) and 7 features (1A, 1B, 2A, 2B, 2C, 3A and 3B). First functionality is implemented using two alternative features, whereas second one has three alternative features and third one comprises two alternative features.

3.1.4 Analytical Modeling. In [1], we use Markov Decision Process (MDP) as the analytical model. There are several reasons behind our choice to refer MDP. MDPs are simply conventional state-transition systems with annotations on transitions and software engineers use state-transition systems very frequently in practice while designing and analysing the software systems. In addition, the initial model of MDP can be expressed as an activity diagram or any other control-flow model of the application and there are several approaches available in literature to support automatic model-to-model transformation from activity diagrams to MDP. An example is the ATOP [19] tool.
The transformation process comprises three sequential steps.

(1) It first translates activity diagrams of each abstract functionality into simple MDP with an initial state, a final state and as many as intermediate states as the number of alternative features associated with the abstract functionality.

Table 1. : Feature Priority-Adaptability Mapping

Input (Feature Prioritization)		Output (Adaptability Mapping)	
Priority Type	**Feature Group**	**Unavailabilty**	**Degradation**
F_C	F_R		Yes
F_I	F_R	Yes	Yes
	F_O	Yes	
F_U	F_O	Yes	

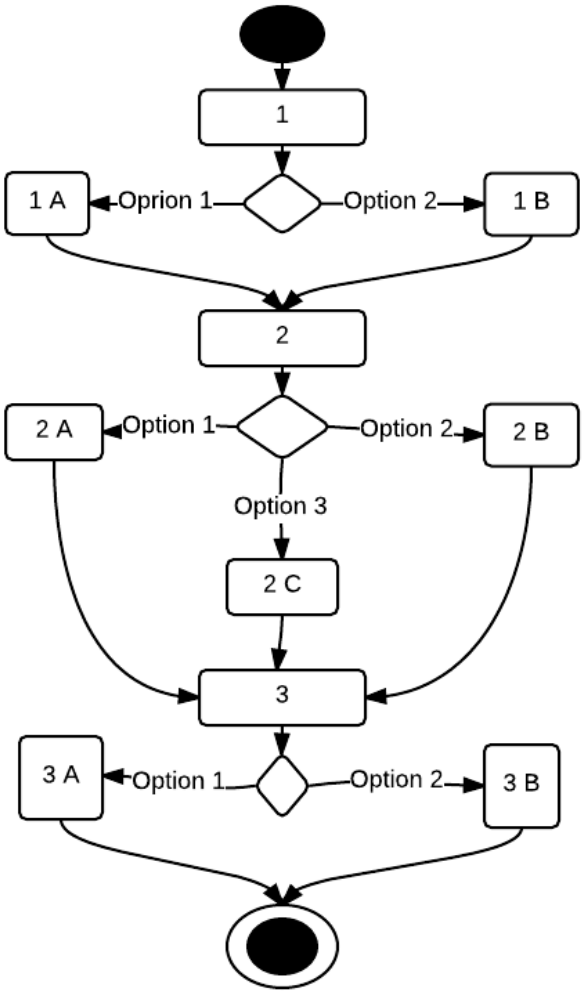

Fig. 1: Extended Activity Diagram

(2) Next, it adds non-deterministic transition from the initial state to each intermediate state, in turn connected with the final state (illustrated in Figure 2). These non-deterministic transitions are labeled with contextual requirements.

(3) Finally, the resulting MDPs are merged into a single MDP that represents the transition of the overall Activity Diagram.

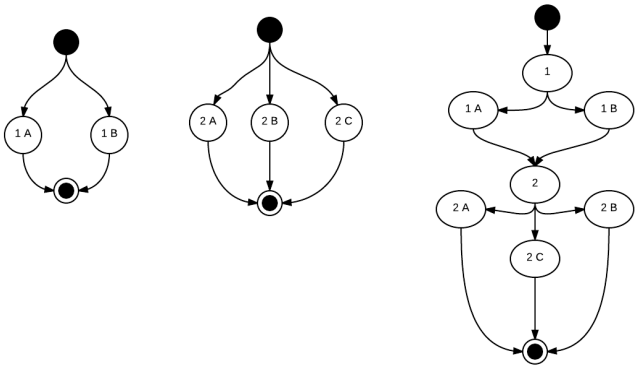

Fig. 2: MDP Translation Process

Table 2. : Alternative Features

Functionality	Contextual Factor: Min Requirements	Features
Contact with doctors	Network: Strong Battery: Full	Real time video
	Network: Average Battery: Medium	Real time voice
	Network: Poor Battery: Low	Real time chat
Locate Nearest Emergency	Network: Average Battery: Medium Location: GPS	Display Map
	Network: Battery: Low Location:	Display Text
Receive Prescription	Network: Medium	Download Detail
	Network: Poor	Show Summary

3.2 Runtime

The overall MDP generated at design time is used to verify the the runtime condition of the system against the minimum context requirements. This verification process is done using a *stochastic model checker* [20] which can check runtime contextual properties against the minimum requirements and generates an alert if it finds any divergence from the ideal requirements. This helps to determine the adaptation decision proactively as the verification process is being performed continuously and alert comes before execution of the feature. For implementing this task properly, we present an adaptation manager which is composed of three modules: Observer, Interpreter and Executor.

3.2.1 Observer. Observer is programmed to collect data regarding to operational contexts. It relies on the hand-held device operating system to collect those information. All popular operating systems for smart phones and tablets offer a set of public APIs for developers to collect sensed information and make use of them. This module should be programmed to run continuously as it can use the APIs offered by the device operating system and collect context information continuously.

3.2.2 Interpreter. Interpreter is responsible for execution of analytical model using *stochastic model checker*. It continuously navigates through the each state of the analytical model and checks each of the contextual factor against the collected context data. Interpreter executes following calculation and generates an alert if necessary:

—For single-valued contexts, it simply executes a conditional statements that returns true if the context is available and returns false if the context is unavailable.

—For threshold-based contexts, at first it calculates the summation of minimum values of each context that is required to reach the final state. Then, it checks the summation against the available context. Finally, it returns true if available context is greater than the summation of minimum values, otherwise returns false.

Returning a *false* by interpreter signifies an alert should be generated for the corresponding feature.

3.2.3 Executor. This module is responsible for executing the feature unavailability or degradation mechanism. It switches the status of the features depending on the contextual criteria. For this purpose, we use *Bundle Manager Algorithm* of [10]. *Bundle Manager Algorithm* uses threads to manage different states depending on contextual criteria. If 'Bundle Manager' receives any alert from the interpreter, it starts a new thread with updated 'available feature set' and destroys the running thread. To ensure uninterrupted service, it locks the running thread if it finds any feature running. That means, deactivation of an operational feature will be postponed until the operation is completed. By doing this, it maintains the runtime consistency.

Next, each of the features modeled by generating a template for their concrete implementations.

Listing 2: Feature modeling of Locate nearest hospitals

```
@Functionality (name = "
    LocateNearestHospitals")

@Feature (name = "LocateNearestHospitalByMap
    ")
@Level (metrics = {"Network", "Battery", "
    GPS"}, values = {2,2,1})
public String automaticLocationIdentifier (
    String position) {
   // invoke GPS service
}

public String displayMap () {
   // invoke Map service
}

@Feature (name = "
    LocateNearestHospitalByText" )
@Level (metrics = {"Network", "Battery", "
    GPS"}, values = {0,1,0})
public String mannualLocationIdentifier () {
   // ask user to insert his/her current
       location
}

public String displayText (String location)
    {
   // show address querying database
}
```

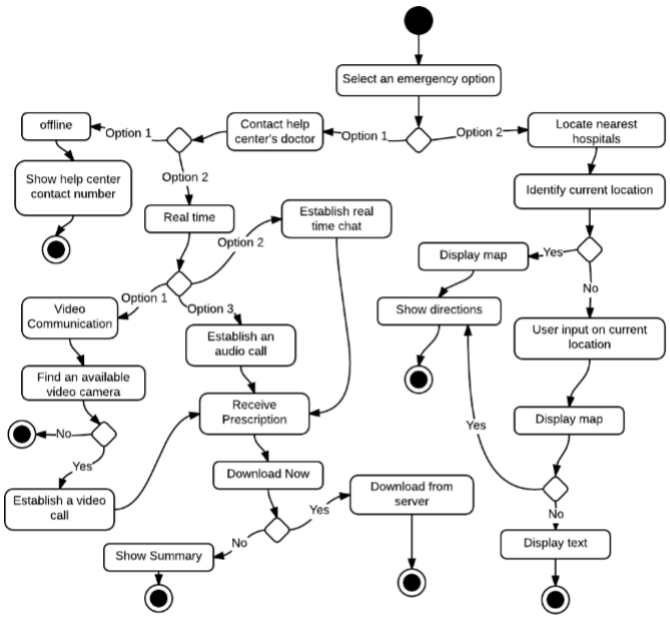

Fig. 3: Overall Activity Diagram of e-health Care

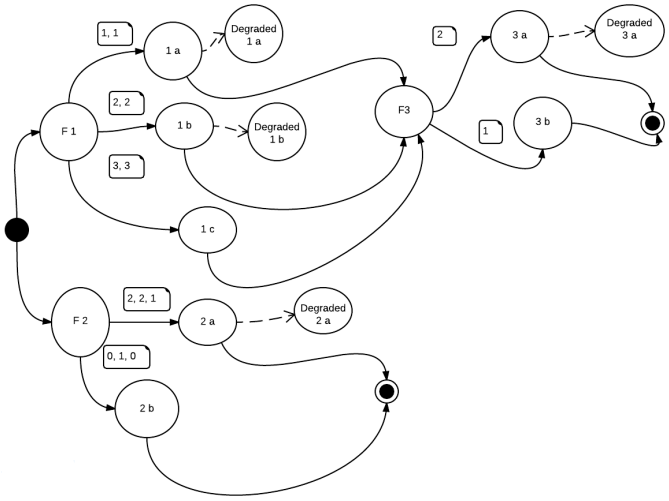

Fig. 4: Overall MDP of e-health Care

4. EXPERIMENT

We design a mobile distributed emergency response system that is intended to aid during medical emergencies. We named it 'e-Health Care'. Though this application may offer more services in practical, for simplicity we limit those to following ones:

(1) Contact with the help center doctors
(2) Receive Prescription
(3) Locate nearest hospitals

We implemented e-health Care application using two different approach. The first approach follows traditional application development methodology which offers a static set of features. We named it as *Non-Adaptive application*. In this application all the features were available throughout the life span of the application. The second approach followed the proposed framework which updates available feature set dynamically at runtime depending on contextual factors. The identified alternative feature sets are illustrated in Table 2. An example of the alternative implementations is shown in 2. The overall activity diagram of e-health Care is illustrated in Fig. 3. Using the activity diagrams, the corresponding MDPs are generated. Firsts, we generated three states for three functionalities annotating F1, F2 and F3. Here, F1 signifies *Contact with doctors* functionality, F2 is *Locate nearest hospitals* functionality and F3 signifies *Receive prescription* functionality. Then, we generated different states for each of the features and augmented an automaton by creating paths from the states of their corresponding abstract functionalities. Each of the path is annotated with the minimum context requirements needed to execute the feature. The overall MDP of e-health Care is illustrated in Fig. 4
During our experiments, the test have been conducted using LG G2 mobile device with Android version 5.1.1.

4.1 Context-Awareness Testing

To evaluate the usability increased by our framework, we examined the battery saving increased by our framework. First, we ran the non-adaptive version of the application. In this case, we attempted to use all the features of the application repeatedly until the complete shutdown of the device. We repeated the test several times starting at different battery level each time.

On the second phase, we ran the adaptive version of the application. The test was done by running the test application under same conditions that were used in non-adaptive version. In this case, if the service of the feature is available, we use the feature, otherwise we move to next feature, and so on. The adaptive application has increased on an average 30.46% battery life than non-adaptive application. The result is illustrated in Figure 5.

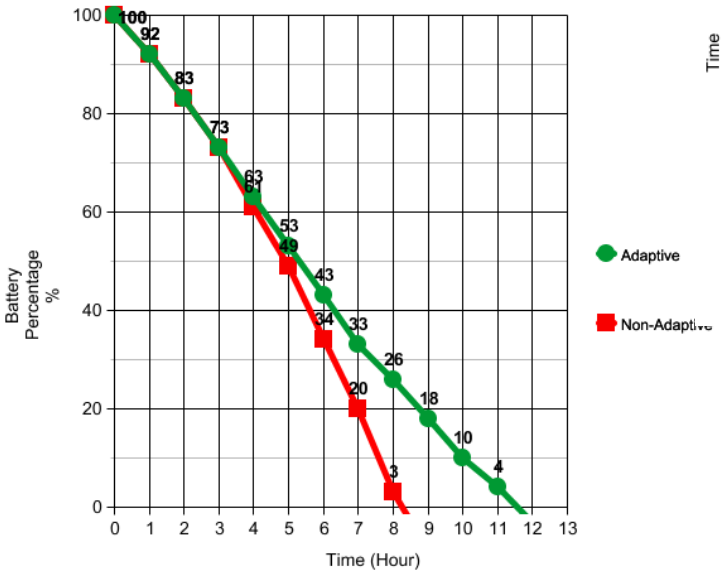

Fig. 5: Battery Saving Provided by the Proposed Framework

Figure 5 shows the proposed framework extends the lifetime of the battery more than 3 hours. In other words, user can get a considerable additional time while using any application that implements the proposed framework. A point to be noted here that, until 70% of battery, the adaptive and non-adaptive applications will have the same battery consumption. However, things will change when battery reaches 70% as the adaptive application will move to change its configuration biased towards energy saving.

4.2 Runtime Overhead Testing

Since the framework requires additional runtime computation, we verified the overhead imposed by the framework. First, we measure the battery usage to enable and disable threads. The experiment result is shown in Fig. 6. From Fig. 6, we can see that activity of stopping and activating features had no significant impact on the battery, only 1.5% battery was consumed when we deactivated 7 features at once. In real world, it is a very rare case that 7 features are affected by context changes simultaneously.
We also measure the time to enable and disable the features. The result is illustrated in Fig. 7. Fig. 7 shows that the time needed to activate and stop bundles is in the range of milliseconds and it grows linearly with the number of features. The more features stop and activate at once, the more time needed to achieve this task. Having said that, activating and deactivating 7 features at a time will need an average of 0.6s, which is very insignificant according to user experience.

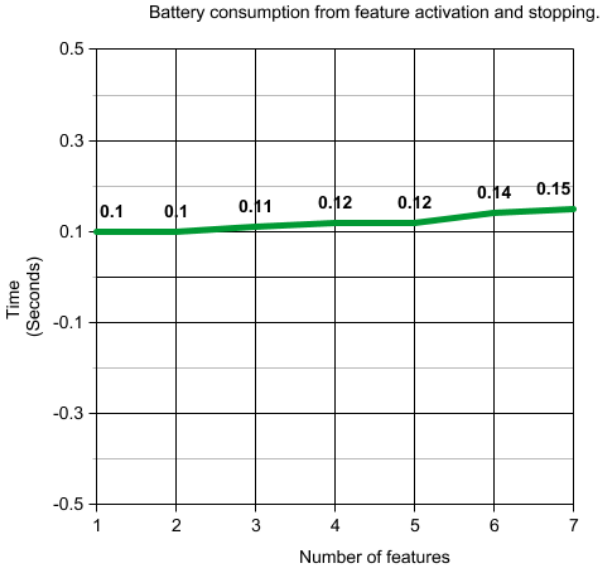

Fig. 6: Battery Consumption from Feature Activation and Stopping

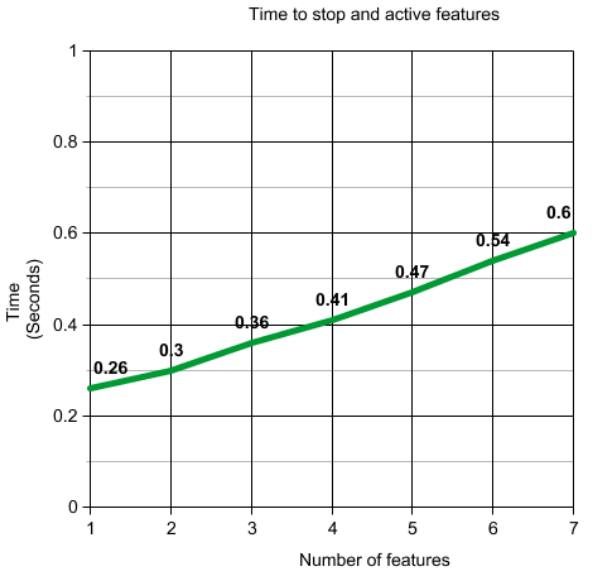

Fig. 7: Time to Stop and Active Feature

5. CONCLUSION

This paper identifies the necessity of a light-weight and proactive adaptive framework for mobile applications and proposes an approach that supports design, development and runtime management of the self-adaptation mechanism in mobile applications. It uses the requirements models of the model-driven software development framework to make the adaptation decision proactively. To minimize the complexity of the adaptation strategy, it uses dynamic selection of mutually exclusive features instead of architectural reconfiguration. The outcome of this adaptive framework is really

significant while the overhead imposed by the framework is negligible according to user experience.

6. REFERENCES

[1] M. Shafiuzzaman, N. Nahar, R. Rahman, "A Proactive Approach for Context-Aware Self-Adaptive Mobile Applications to Ensure Quality of Service" in *International Conference on Computer and Information Technology (ICCIT)*, Dhaka, Bangladesh, 2015.

[2] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," *Software Engineering Institute*, 1990.

[3] B. Schilit, N. Adams, R. Want, "Context-Aware Computing Applications," *Proceedings of the International Workshop Mobile Computing Systems and Applications*, 1994.

[4] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, P. Steggles, "Towards a Better Understanding of Context and Context-Awareness," *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, 1999.

[5] J. Bezivin., "Model driven engineering: An emerging technical space in Generative and Transformational Techniques," *Software Engineering (GTTSE)*, volume 4143 of LNCS, pages 36-64. Springer, 2006.

[6] J. Kramer, J. Magee, "Self-Managed Systems: an Architectural Challenge," *International Conference on Software Engineering*, Minneapolis, Minnesota, 2007.

[7] C.Ghezzi, L.S.Pinto, P.Spoletini, G.Tamburrelli, "Managing non-functional uncertainty via model-driven adaptivity," *International Conference on SoftwareEngineering*,2013.

[8] D. Cooray, E. Kouroshfar, S. Malek, R. Roshandel, "Proactive Self-Adaptation for Improving the Reliability of Mission-Critical, Embedded, and Mobile Software," *IEEE Transactions on Software Engineering*, 2013.

[9] N. Esfahani, A. Elkhodary, S Malek, "A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems," *IEEE Transactions on Software Engineering*, 2013.

[10] R. Mizouni, M. A. Matar, Z. A. Mahmoud, S. Alzahmi, A. Salah, "A framework for context-aware self-adaptive mobile applications SPL," *Expert Systems with Applications*, Volume 41 Issue 16, November, 2014. Pages 7549-7564 2013.

[11] M. N. Bennani, D. A. Menasce, "Assessing the Robustness of Self-Managing Computer Systems under Highly Variable Workloads," *Int'l Conf. on Autonomic Computing,* New York, 2004, pp. 62-69.

[12] T. Ryutov, L. Zhou, C. Neuman, T. Leithead, K. E. Seamons, "Adaptive trust negotiation and access control," *ACM Symp. on Access control models and technologies, Stockholm, Sweden,* 2005, pp. 139-146.

[13] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice,* Wiley, 2009.

[14] D. E. Perry, A. L. Wolf, "Foundations for the study of software architecture," *Softw. Eng. Notes*, vol. 17, no. 4, pp. 40âĂŞ52, Oct. 1992.

[15] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257-286, Feb. 1989.

[16] D. Gross and C. M. Harris, *Fundamentals of queueing theory (2nd ed.).* John Wiley and Sons, Inc., 1985.

[17] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, "Dynamic software product lines," *IEEE Computer,* vol. 41, no. 4, pp. 93- 95, 2008.

[18] A. Filieri, C. Ghezzi, G. Tamburrelli, "A Formal Approach to Adaptive Software: Continuous Assurance of Non-Functional Requirements," *Formal Aspects of Computing*, 2012.

[19] S. Gallotti, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Quality prediction of service compositions through probabilistic model checking" *Proc. of the 4th International Conference on the Quality of Software Architectures*, Karlsruhe, Germany, 2008.

[20] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "Prism: A tool for automatic verification of probabilistic systems," *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2006.

[21] D. A. Menasce, J. M. Ewing, H. Gomaa, S. Malek, and J. P. Sousa, "A framework for utility-based service oriented design in SASSY," *Joint WOSP/SIPEW IntâĂŹl Conf. on Performance engineering*, San Jose, California, 2010, pp. 27-36.

[22] S. Malek, N. E. Beckman, M. Mikic-Rakic, and N. Medvidovic, "A Framework for Ensuring and Improving Dependability in Highly Distributed Systems," *Wrkshp. on Architecting Dependable Systems, Florence*, Italy, 2004, pp. 173-193.

[23] R. Rahman, A. S. Ami and K. Sakib, "MobileMonkey - A Contextual Stress Testing Framework for Android Application," *International Journal of Computer Applications*, 172(9):1-7, August 2017

[24] B. Regnell, L. Karlsson, and M. Host, "An analytical model for requirements selection quality evaluation in product software development." *In Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 254-263, 2003.

[25] E Bagheri, M Asadi, D Gasevic, S Soltani, "Stratified analytic hierarchy process: Prioritization and selection of software features," *Software Product Lines: Going Beyond*, 2010 - Springer.