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

Rabeb Mizouni(1), Mohammad Abu Matar(2), Zaid Al Mahmoud(2), Salwa Al Zahmi(2), Aziz Salah(3)

(1) Department of ECE, Khalifa University, Abu Dhabi, UAE
(2) Etisalat British Telecom Innovation Center Abu Dhabi, UAE
(3) Computer Science department, UQAM

{rabebmizouni, mohammad.abu-matar, zaid.almahmoud, salwa.alzahmi}@kustar.ac.ae; salah.aziz@uqam.ca

Abstract—Mobile Applications are rapidly emerging as a convenient medium for using a variety of services. Over time and with the high penetration of smartphones in society, self-adaptation has become an essential capability required by mobile application users. In an ideal scenario, an application is required to adjust its behavior according to the current context of its use. This raises the challenge in mobile computing towards the design and development of applications that sense and react to contextual changes to provide a value-added user experience. In its general sense, context information can relate to the environment, the user, or the device status. In this paper, we propose a novel framework for building context aware and adaptive mobile applications. Based on feature modeling and Software Product Lines (SPL) concepts, this framework guides the modeling of adaptability at design time and supports context awareness and adaptability at runtime. In the core of the approach, is a feature meta-model that incorporates, in addition to SPL concepts, application feature priorities to drive the adaptability. A tool, based on that feature model, is presented to model the mobile application features and to derive the SPL members. A mobile framework, built on top of OSGI framework to dynamically adapt the application at runtime is also described.

Keywords—Mobile devices, SPL, multi-view variability model, feature priority, runtime adaptability

1. INTRODUCTION

The rapid technological advances in mobile devices have considerably contributed to the widespread of mobile computing. Information is being continuously available to users anytime and anywhere, a fact that increased the need to build applications that sense and react to the contextual change of their users.

The mobile context aware computing paradigm focuses on building applications that can take advantage of contextual information such as user location, current time of day, user profile, and connectivity to improve the overall user experience [1]. Dey and Abowd [2] defined the context as “any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves”. This information needs first to be collected, analyzed, and finally used to adapt the mobile applications.

With the technological advances of today’s hand-held devices, collecting information of the context is no longer an issue, however the type of behavior to adopt is still an open problem [3]. Such adaptation needs to serve several goals determined by the nature of the application, the nature of the context, and the preferences of the user. In some applications, the adaptation can be made to maintain a certain quality of service the user is requiring, in others the adaptation can be made to offer a personalized service to the user such as location-based services, and even to provide user personal suggestions based on the recognized activity and context [4]. For instance, the battery level of the handheld device may be a critical decision parameter in changing the behavior of an application if the goal of such adaptation is to extend the lifespan of the battery and hence the device connectivity.

In order to build adaptive context-aware applications, several solutions were proposed [5-9]. However, most of them do not propose a systematic way to develop adaptive mobile applications throughout the software development lifecycle. To manage variability in adaptive systems, many [10-12] advocate the use of Software Product Line (SPL) techniques due to the complementary benefits of both concepts. Software Product Lines (SPL) aim at constructing systems based on a family of applications that share common
functionality, but each member has some variable functionality [13]. The main goal of SPL is the agile and rapid development of family members by using reusable assets from all phases of the development lifecycle.

In this paper, we propose to use SPL techniques to promote dynamic adaptation of mobile applications. First, we categorize the adaptability that a mobile adaptive application could exhibit to two types: Feature Availability and Feature Degradation. Feature Availability is an adaptability that changes the availability of the features at runtime. According to the context change, some features may be deactivated, others activated to better serve the current user context. As a consequence, the user may observe a change in the application GUI. Features that cannot be offered in the sensed context will be disabled (shaded). Only features that can be handled by the sensed context are offered to the user. Feature Degradation is an adaptability that changes the behavior of a given feature. This behavior may change to adapt to the sensed context. In order to implement these adaptability types, we present a framework for modeling and designing context-aware adaptive mobile applications. We propose a multi-view SPL based variability approach which incorporates a modeling approach to easily adapt mobile applications to context changes. In our framework, the adaptation to context changes corresponds to product derivation in SPL.

To help and guide adaptability, a new feature model is defined. This model extends the feature model in [12] with three feature groups, namely critical, important, and useful, and a context of use associated with each feature, denoted by ContextReq. The latter represents the minimal resources needed, as estimated by the modeler, to run such variation. Critical feature group represents all the features that must be included in all SPL members of the adaptive mobile application, while important and useful feature groups may be included in some SPL members and excluded in others. This selection is in fact context driven. Moreover, to help achieving runtime adaptability, new feature modeling is proposed for each of the added groups. Modeling Critical and important feature groups require the distinction of two different groups: the kernel group and the optional group. Kernel group represents the part of the feature that needs to be incorporated in all derived SPL family members. Optional groups model the variability part among family members. To the opposite, a Useful feature group is modeled with alternatives that may be included in SPL members. At the end of the design time, different feature models for the various SPL members, each associated with a context of use, are created. These models feed the framework implemented in the mobile device. In addition to the traditional context acquisition module, the framework incorporates a context decision module that assesses contextual information and decides about the adaptation to perform based on the defined feature model.

As a proof of concept, a framework built based on the proposed multi-view variability model that helps the generation of mobile application SPL, is presented. In addition to handling the changes of the application at design time, runtime mobile application adaptation is realized using OSGi [14] bundles deployment. An m-health application for the Android platform that is self-adaptive and context-aware is presented.

To summarize, the contribution of this paper is threefold:

1. Propose a new multi-view variability model for modeling adaptive mobile applications that promotes adaptability
2. Incorporate SPL techniques to derive adaptive mobile application
3. Propose a framework for mobile devices to dynamically adapt to context changes based on OSGi bundle deployment.

The rest of this paper is structured as follows. Section 2 describes our motivating scenario. Section 3 lists some approaches that have proposed solutions to context aware mobile computing. Section 4 introduces adaptability types that we introduce in a running mobile application. Section 5 describes the multi-view variability model. Section 6 presents the framework for context-aware self-adaptive mobile applications SPL. Section 7 presents the application we built as proof of concept of our framework. Section 8 concludes the paper and outlines some future directions.

2. Some facts about Mobile Computing

Recent statistics [15] show that in 2013 the number of smartphones in use worldwide has broken the 1 billion mark with a global smartphone penetration of 16.7 percent, a number that is expected to increase during the coming years. According to the same source, the number of smartphone subscriptions is
predicted to grow to 3.3 billion in 2018. In addition, 195.4 million tablets were sold in 2013 and it is believed that by 2016, smartphones and tablets will outsell feature phones and laptops by a considerable margin, a fact that will definitely change the face of ubiquitous computing. With this significant penetration of smartphones and tablets, millions of mobile applications are installed and are in use anywhere and anytime by millions of users. Generally speaking, mobile applications are software applications designed to run on a handheld device such as email clients and games. Development of these applications requires the consideration of many factors such as the power constraints and the screen size of the handheld device.

The mobility of users represents another challenge for the development of mobile applications. While using her/his phone on the move, the environment of the user keeps changing constantly. To cope with context change, mobile applications need to be self-adaptive in the sense that they are judiciously instrumented to make efficient and suitable use of the mobile environment resources and context. A self-adaptive software system [16] is one that can modify itself at run-time due to changes in its context; unfortunately it is not the case for most of mobile applications currently in use. As mentioned in [17], most of the applications deployed in users’ smart phones and tablets have a static behavior. They show only slight changes to contextual changes and even when the battery reaches critical levels, or the network speed is low, those applications would offer the same features and would exhibit almost the same exact behavior; WhatsApp is one these applications. Up to the time of writing (January 2014), WhatsApp has 430 million users and offers to its users the possibility to create groups, send each other unlimited images, video and audio media messages. These features are in fact battery and network resources intensive. However, WhatsApp does not offer any sort of dynamic adaptation that could be based on the current device status and user context, except an adaptation to network availability. The mobile client will make all features unavailable if the device is not connected to any network (3G/4G or WIFI). However, regardless of the battery status, once the device is connected, all features are made available and users can start a voice conversion without getting any notice even though such action will consume the remaining battery and cause most probably the disconnection of the device. Facebook also exhibits the same behavior (as presented in [17]) and do not deactivate features to efficiently use the available resources. Though, YouTube shows some context awareness and degrades the quality of the media if low level battery detected. The disconnection of the handheld device in such situations could be avoided if mobile applications are more context-aware.

The idea featured in this paper examines the problem of adapting mobile applications to context changes by adjusting the running application [1]. Questions this work tries to address are:
1. Should mobile applications offer the same features to their users even when their context is continuously changing? Can an application disable user services progressively and enable others (such location-based services) to adapt to context changes?
2. How can we efficiently design such context-aware adaptive applications so that variability is managed throughout the development process?
3. How and using what mechanism can mobile applications self-adapt to context changes at runtime?

3. Related work
With the widespread of mobile computing, many research projects tackled the challenge of improving the quality of experience of mobile applications [18] through adaptation according to the context of use [5-9]. Context-aware systems sense the context, analyze it, and adapt to provide relevant information and services to the user. Nowadays, many platforms for developing context-aware mobile applications are available [19-21]. Recent survey papers [22, 23] are already published to overview the current state of the art of context aware mobile application development. These solutions can be divided to many categories, depending on the type of the adaptation proposed. In what follows we give an overview of the main trends.

Some research papers focus on developing middleware solutions [24] [17] [18] to implement context-awareness. CAPPUCINO [25] is a platform for executing context-aware web services in ubiquitous environments. The middleware is built as an autonomic control loop that deals with dynamic adaptation. This platform ensures the adaptation in service oriented ubiquitous systems. CAPPUCINO is not a device centric approach where the device decides the adaptation to perform. All the decisions are in fact made at the server side and then deployed within the mobile device. In line with CAPPUCINO, CARISMA [26] is
also a middleware for constructing context aware adaptive applications. Services and adaptation are installed and uninstalled on the fly. Changes of the context trigger application profiles’ changes. Recently, in [27] presented also a context-aware middleware for service surveillance video that is privacy aware. Joeng et al. in [18] propose a mobile application streaming service in which the server side infrastructure does most of the job and customizes the screen display for the mobile device. This approach assumes a permanent availability of a connection to mobile web services.

Other research projects focus on developing self-adaptive mobile applications that make adaptation decisions on the fly at the device side but most of them are domain oriented. Various applications are developed to use the current location of the user as a main factor to perform a geo-location based adaptation [28, 29]. In [4], the authors presented a framework to detect the current context and activity of the user on the fly then offer a personalized content. In [30], authors propose an approach for developing mobile applications with separation of concerns: functional and adaptive behavior in an ad-hoc network. The authors however did not study specific aspects of mobile application. In [31], the authors propose a novel on-device sensor management strategy and a set of trajectory updating protocols which intelligently determine when to sample different sensors (accelerometer, compass and GPS) and when data should be simplified and sent to a remote server. The system is configurable with regards to accuracy requirements and provides a unified framework for both position and trajectory tracking.

Since most of the approaches would realize the adaptation through conditional statement injected in the source code of the system either at the architectural level or system component level which often leads to create an application that is hard to read and maintain. Context-Oriented Programming (COP) [32] has been introduced. It is a novel paradigm that provides language to modularize the behavioral concerns “variation activation” within the system and to dynamically activate them during the program execution to realize the desired adaptation to provide a language –level abstraction approach for managing context aware system [33] [34]. For instance, JCop [35] language has been introduced to provide a more declarative approach by pointcut-like adaptation rules for mobile application development. COP is similar to feature oriented programming and dynamic software product line approaches that we adopt in this work. They share the same notion of using variation point within the original system. However, COP method is specifically focusing on Language-level aspect without providing a systematic mechanism of managing those adaptation variants at the design time compare to SPL techniques.

Finally, it is to mention that rule based methods [36] have been utilized in context-aware systems [37]. Rules are used as representations of models and as support for reasoning. In recent work[38], the authors proposed a rule-based context reasoning platform for context-aware and adaptive mobile applications. An inference engine responsible for performing reasoning has been implemented and deployed in the mobile device. Reasoning techniques can be data drive or goal driven. In our approach, we apply a systematic mechanism to define those rules based on feature model and SPL variability. These rules are provided to a context manager module, implemented at the mobile framework, responsible for reasoning and deciding on the adaptation actions.

As mentioned in [22], many of the approaches in adaptive mobile computing are based on end-user programming which enables adding functionality that has not been anticipated by the system designer, a solution that is not suitable in all domains. Considerable effort is made to propose solutions for self-adaptive application, however challenges to design and have systematic approach to build those application is still open. In this paper, we present an approach to help integrating adaptation as part of the design process in order to end up with context-aware mobile application. Using SPL techniques, our objective is to define a systematic approach and framework that based on feature selection will help realizing the adaptability the designer is targeting.

### 4. Dynamic Adaptability

Any mobile application can be seen as a set of features that are manipulating a set of data in order to answer some user needs. Those features are implemented to enable the set of services the application is offering. We define context as the circumstances, situations or environment in which a particular system exists [39]. Practically, the context of a mobile application is assessed at runtime. To efficiently adapt mobile
applications with respect to context changes, we define two adaptability types, namely Feature Availability, and Feature Degradation. In what follows, we define each of them.

4.1. Feature Availability
A feature is an abstraction of a capability provided by the system [24]. As argued in [24], adopting a feature-based model of adaptation helps the development of self-adaptive applications in general. In any software application, not all features have equal standing. If the context changes, it may be wise to not satisfy a non-critical feature if it helps to continue satisfying critical features [40]. Feature Availability is meant to help achieving this goal. It is defined as the modification of the availability of the application features to answer the changes the context is undergoing.

While, at runtime, static mobile applications offer to the users all the set of features they propose, Feature Availability will allow to disable some features and enable others when an assessment of the context is performed. As a result of this adaptation, the user will not necessarily deal with a similar pool of features each time s/he launches the application. For instance, if the battery level of the device is low, non-critical features that are battery intensive may be disabled.

Defining rules for disabling features according to contextual changes for a given mobile application is a challenging task. In fact, designers have constraints on what to make unavailable. Some judicious criteria for the choice of features to disable are feature priority and feature resource needs.

1. Prioritization relies on giving more weight to the most important features. A common approach to prioritization is to group features into three priority categories [41]: critical, important, and useful. This approach is commonly used in project development to make tradeoffs during project plan definition according to the available resources for building the product. We extend its use to application feature model definition. Using the same semantic, critical and important features define the core of the application services. Useful features are seen as nice-to-have. This prioritization helps the designer to perform trade-offs when it comes to dynamically adapting an application to a given context change. In case of resources insufficiency, features that have high priority are kept available while those with lower priority are progressively made unavailable according to altered context.

2. Resources consumption estimation is the second criterion for feature selection. Obviously, when two features have the same priority level, the decision would be based on the estimation of the minimal resources needed to deploy a feature. When the battery of a given device is low, feature implementations that consume less energy would be those deployed.

Feature Availability may require Graphical User Interface (GUI) layout change at runtime. As an example, geo-specific features could be disabled once the location of a user changes. During this work, we assume that users are aware of such fact and agreed to have dynamic GUI layout changes as a result of the self-adaptive capability of the mobile application. At the application implementation level, such option is offered to the user as part of the application configuration and it indicates whether or not she/he accepts GUI flexibility. Obviously, once the user chooses to disable this option, the application loses its self-adaptive capability. The default configuration is hence deployed.

4.2. Feature Degradation
As features have different priority levels, Feature Availability adaptation cannot be applied to any feature of the system. Reasonably, even in extreme cases of context changes, critical features that represent the core services of the application cannot be completely disabled, since this would jeopardize the integrity of application goals and usability. A balance between usefulness and context adaptability has to be established and some features need to remain available to assure the minimal services needed. We propose adapting the internal behavior of application features. In fact, different versions of the same feature are implemented. The version to call is correlated with the state of current context. The developed application would exhibit context-aware behavior that is manifested through the self-adaptation of the available feature behavior.

We note that nowadays, mobile applications are dealing with different data formats and sizes. In addition, recent works promoted content adaptation [4]. We consider this as a type of internal feature behavior adaptation. According to the battery level, and network status, the application should choose the right format to deal with when the application moves to a configuration biased towards energy saving.
4.3. Running Example: Adaptive m-Health Client for Doctors on the Move

A running example is used to show the applicability of our approach. As home patient care is increasing, doctors and nurses are more and more on the move. They need applications to help accessing patient information while they are visiting them. In addition, they need to update patients’ records. Recently, many researchers [42, 43] focused on providing mobile applications to help them in this task. Because of doctors’ mobility, the context of use of these applications is continuously changing. In addition to network-related changes, smartphones/tablets, where the application is deployed, also undergo some changes like battery drainage, or network disconnection to cite a few.

To improve the experience of doctors, we propose to develop an adaptive and context-aware healthcare doctor mobile client. It is an application that helps doctors on the move to have access to patients’ files, report medical conditions, prepare for intervention and advise hospital and emergencies about patient needs and arrival. Summary of the mobile client features, their respective priority levels are shown in Table 1.

5. Multi-View Model for Mobile Application SPL

In this section, we give an overview about SPL. Then we present the feature model we propose to guide the design and the deployment of adaptive mobile applications. Finally, we define the multi-view variability meta-model implemented in the framework described in the next session.

5.1. Software Product Lines & Feature Modeling

Software Product Lines (SPL) are families of software systems that have some common functionality and some variable functionality [12]. The main goal of SPL is the reuse-driven development of SPL member applications by using reusable assets from all phases of the development life cycle. An essential modeling phase in Software Product Lines Engineering (SPLE) is Commonality and Variability Analysis (CVA) where the common and variable features of SPL member applications are determined. CVA is commonly expressed in feature models based on the SPL varying requirements.

SPL development consists of two main processes [12]: a) Domain Engineering. A SPL multiple-view model, SPL architecture, and reusable components are developed and stored in the SPL reuse library. b) Application Engineering. The application developer selects the required features for the individual SPL member. Given the features, the SPL model and architecture are adapted and tailored to derive the application architecture.

<table>
<thead>
<tr>
<th>Feature Priority</th>
<th>Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>F1</td>
<td>View Patient File</td>
</tr>
<tr>
<td></td>
<td>F2</td>
<td>Search in file</td>
</tr>
<tr>
<td></td>
<td>F3</td>
<td>Search medicine</td>
</tr>
<tr>
<td></td>
<td>F4</td>
<td>Search by date</td>
</tr>
<tr>
<td></td>
<td>F5</td>
<td>Search by illness</td>
</tr>
<tr>
<td></td>
<td>F6</td>
<td>Report</td>
</tr>
<tr>
<td>Important</td>
<td>F7</td>
<td>Identify the nearest emergency</td>
</tr>
<tr>
<td></td>
<td>F8</td>
<td>Identify the district emergency</td>
</tr>
<tr>
<td>Useful</td>
<td>F9</td>
<td>Planning intervention</td>
</tr>
<tr>
<td></td>
<td>F10</td>
<td>Real-time assistance (vocal, textual)</td>
</tr>
</tbody>
</table>

Table 1: m-Health Client for Doctors on the move: Feature Description
5.2. Feature View
Feature modeling is rooted in the seminal work of Kang et al. [39] in the Feature Oriented Domain Analysis (FODA) method. It is the activity of identifying externally visible characteristics of products in terms of features and organizing them into a feature model. In SPL approach, feature models are used to express and manage similarities and differences among different family members in a product line.
Features are analyzed and categorized as common, optional, or alternative. Common features among products in a product line are mandatory or kernel features, while different features among them may be optional or alternative features. Related features can be grouped into feature groups, which constraint how features are used by a product of a product line. A feature group is specialized to “zero-or-more-of”, “at-least-one-of”, “exactly-one-of”, or “zero-or-one-of” [12].

According to the adaptability types identified in the previous section, the feature model should:
1) Clearly identify feature priorities (critical, important, and useful), and resources required as they are the two main criteria for removing/changing features when dynamically adapting the mobile application as adaptation to a context change corresponds to member derivation in SPL.
2) Clearly distinguish two levels of feature granularity: a) the feature priority level to accommodate the external feature adaptability and b) the feature behavior description level where the kernel part of a feature is clearly distinguished to guide the internal feature behavior adaptation.

5.3. Multi-View Variability Meta-Model
In this section we present the multi-view variability meta-model to represent and design adaptive mobile SPL applications. Meta-models express the elements of a domain, the relationships between these elements, and the constraints that govern their structure and behavior.
In addition to the feature view, we propose to associate with it four more views, namely the provisioning view to specify the operating system needs, the platform view to specify the platform requirements to run such feature, the composite view to derive the components, and the runtime adaptability view to model the runtime adaptability of the mobile application. In summary, the meta model encapsulates:

- The feature View, as described previously,
- The provisioning view -- this view describes the hardware and/or the virtualized infrastructure associated with a certain SPL mobile member application. This view is modeled by UML Class diagrams extended with <<infra>> stereotypes. This view consists of an Infrastructure element which consists of Server, CPU, Memory, Storage, and Network elements (Fig. 1).
- The platform view -- this view describes the solution stacks of the specific SPL mobile apps. We use UML Class elements extended with <<platform>> stereotypes. The main element in this view is the Solution Stack element, which consists of OS, App Server, Database, and Web Language elements (Fig. 1).
- The runtime adaptability view : it specifies the elements of the context that will derive the runtime adaptability. In this view, we distinguish between the device related element like the battery requirements, the CPU requirements to cite a few and the environment related elements like the...
user location and available networks.

- The composite view: during the component derivation, we distinguish between the kernel composite that contains all the critical feature group and the optional components that contain the optional features.

We extend the feature model in [44] with three new stereotypes which will govern how these groups are considered while member applications are derived. In addition, since our focus is to clearly express variability, it is important to model the context requirement at the feature model. This represents the minimal context requirements needed to run a certain feature. For instance, to post a video in Facebook, the smartphone should be connected via Wi-Fi or 3G/4G. If it is not the case, users cannot post the video. Network connectivity becomes in this case a necessary requirement to the availability of the feature. For this purpose, we extend the feature model with the stereotype <<ContextReq>>.

5.3.1. <<Critical>> Group Modeling
All features in the <<Critical>> group need to be part of all SPL mobile members. However, since we are targeting the two types of adaptability (feature degradation and feature availability), Critical elements are identified by their kernel part, which is a group of mandatory behaviors designers need to include in any member application of the SPL, and some optional behaviors that may augment the kernel part if the context allows it. <<Exactly-one>> group ensures that any generated member will in fact encapsulate exactly one kernel version of each critical feature. Figure 2(a) shows the Patient Report critical feature of our running example. It indicates that reporting text, the kernel part, has two versions: the version that sends all the data to the hospital, which obviously requires a good connection speed as well as a medium to high battery level of the doctor device, and the version that sends some critical data, as specified by the doctor when the network is slow or the doctor’s device runs out of battery. One of these versions needs to be included in any member application of the SPL. The feature has also two options: report images and report video that will be activated only if the context allows it.

5.3.2. <<Important>> Group Modeling
Important features will be under this group. These features are not necessarily part of the kernel; however they are important to the user. It corresponds to <<zero-or-more>> in the feature meta-model. Preferably, all features in the important group are represented in the derived SPL members. However, in order to add more flexibility to the adaptation the application may undergo during the runtime to obey the context changes, <<Important>> features can be disabled. In order to handle such changes at design time, we propose to model important features the same way we model critical features with the only difference that they may not be part of the kernel. They are modeled as <<zero-or-one>> group. During the derivation and according to the context, these features may be all included, some of them included, or in extreme cases, none of them included in the SPL member. Figure 2 (b) shows the specification of Identify Nearest Emergency feature. If this feature is included in the derived member application, then exactly one of the kernel versions, Display text or Display Map, needs to be included. The two others are optional and, similar to <<Critical>> group, they will be part of the derived application only if the current context allows it.

5.3.3. <<Useful>> Group Modeling
Useful features are considered as optional [45]. Consequently, either the current context allows their activation and they will be made available, or the context does not and these features will be progressively disabled. They are modeled by alternatives. Each alternative is associated with the context requirements needed to activate that variation. Unlike the previous groups, <<Useful>> features group does not have a kernel part. Disabling it will not harm the core services of the mobile application. As shown in Figure 3, Plan Intervention feature has three alternatives that have different minimal requirements needed. If the feature will be included in the member application, one of these alternatives is chosen.
5.3.4. **<<ContextReg>>** Modeling

The context requirements description needs to capture (1) *User* requirements/profile/contextual data such as location, (2) *Environment* requirements in terms of type of the operating system the application can be executed on, bandwidth, security level required, type and speed of the network, etc., and finally (3) *Device* requirements in terms of battery, and device capabilities [17]. It represents the minimal resources needed to run such feature or part of a feature. From runtime perspective, **<<ContextReg>>** specifies the minimal context value that can allow activating/selecting such feature.

Each element of the **<<ContextReg>>** can be modeled as static or dynamic. Static elements—requirements that do not change over time—represent any element of the 1) stack solution such as the

---

**TABLE 2: Context Requirements for the m-Health client**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Context Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Speed; Type Battery</td>
</tr>
<tr>
<td>F2</td>
<td>Speed</td>
</tr>
<tr>
<td>F3</td>
<td>Speed</td>
</tr>
<tr>
<td>F4</td>
<td>Speed</td>
</tr>
<tr>
<td>F5</td>
<td>Speed, Type Battery</td>
</tr>
<tr>
<td>F6</td>
<td>Type Battery, Location</td>
</tr>
<tr>
<td>F7</td>
<td>Battery Location</td>
</tr>
<tr>
<td>F8</td>
<td>Speed, Type Battery</td>
</tr>
<tr>
<td>F9</td>
<td>Speed Battery</td>
</tr>
<tr>
<td>F10</td>
<td>Speed, Type Battery</td>
</tr>
</tbody>
</table>
operating system the feature is implemented for; and 2) platform related elements such as CPU and memory requirements. These include hardware affordances, platform capabilities and UI conventions, and the environment in which the feature can be used. Dynamic elements refer to the runtime context such as network availability, speed, status of the battery. They represent the criteria for adaptability. They represent all the elements that change overtime and need monitoring to guide the adaptation.

Back to our m-Health example, we define the <<ContextReq>> of the mobile application by three elements: the battery level (high, low, medium), the speed of the network (high, low, medium), the type of the network (Wi-Fi, 4G), the connectivity (connected, disconnected) and finally the location of the user.

6. FRAMEWORK FOR MOBILE APPLICATION DYNAMIC ADAPTABILITY

Figure 4 shows the framework we are proposing to develop and deploy context-aware adaptive mobile applications using SPL techniques. In this framework, we distinguish between design time modeling that is based on feature selection and runtime adaptation that relies on context collection, analysis, and OSGI bundles activation. In fact, any context change triggers feature selection, which triggers bundle manipulation.

6.1. Feature Priority and Adaptability Mapping

Adapting features comes with cost. Changing available features or adapting their internal behaviors causes the overhead of context acquisition and processing for decision making. To cope with this issue, we propose to define a Priority-Adaptability mapping. It defines for each feature a priority type and the possible adaptability to perform. Table 3 identifies the mapping we are suggesting. <<Kernel>> and <<Exactly-one>> defined in critical features can be subjected to Feature Degradation only, while their <<optional>> part can be subjected to Feature Availability. This rule is enforced by OCL constraints as explained later. As these features are critical, only deactivation of some optional elements is allowed, but never the core of the service. As an example, if we consider a printing service, color printing could be seen as optional and may be deactivated if the device is running out of battery. However, the core of the service, which is printing, will never be deactivated.

Important features may be subjected to Feature Availability and Feature Degradation as well. This gives the designer more flexibility to deactivate important feature as a last resort to adapt to context changes. This flexibility is of great value when it comes to safety critical applications, such as in the case of our running example. In this case, it is still more beneficial to the doctor to have the critical features of the m-health client rather than ending up with a disconnected device because of a drained battery.
Useful features may be subjected to Feature Availability only. As these features are nice to have, either the context allows their activation or not. This mapping will drive the feature selection of the design-time components as we will show next.

6.2. Design Time

SPL members are derived based on feature selection from the feature model. Derivation of the components of the mobile application SPL members is twofold:

6.2.1. Kernel Variation Generation

In the kernel variation generation phase, critical and important features of the SPL are designed, developed, and deployed in the mobile device. This is similar to the ‘Kernel First’ approach in the SPL PLUS method [13] with three fundamental differences: (1) Only $<<$Kernel$>>$ and $<<$Exactly-one$>>$ feature group of critical and important features may be included in the kernel; (2) Many kernel components are generated. Each component includes all the critical features, with variability though on the kernel version to be included on the kernel component. Kernel components do not necessarily include any feature of the $<<$Important$>>$ feature group. It is a decision the designer needs to take. As mentioned previously, deactivating important features may be seen as a last resort to adapt to context changes. The more kernel components’ variation designers derive the more adaptability to context change the application will be able to provide at runtime; and finally (3) Each kernel component is linked with $<<$ContextReq$>>$ that represents the minimal resources needed to run such component. The value of $<<$ContextReq$>>$ of the kernel component is obtained by composition of the context requirements of the selected feature parts included in that component, a task achieved by the Context Requirements Composition Engine as explained next. Unlike [13], each derived kernel is a fully functional instance that offers the core services of the mobile application to users.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature Availability</th>
<th>Feature Degradation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>Kernel/Exactly one Feature Group</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>X</td>
</tr>
<tr>
<td>Important</td>
<td>Kernel/Exactly one Feature Group</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>X</td>
</tr>
<tr>
<td>Useful</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Feature Priority-Adaptability Mapping

At the end of this step, the designer would have derived a set of kernel components. Each SPL member includes exactly one of these kernels. Such dynamicity in kernels allows more flexibility of the adaptability the mobile application may have. At runtime, only one of these kernels is made active to the user. When the context can no more satisfy the context requirements of the running kernel, the running kernel component is stopped to activate another kernel component that the context may handle. In addition, in order to overcome the problem of context satisfiability, at design time, the designer has to identify a default kernel component that will be activated when the current context does not fulfill any of the minimal requirements of all the available kernels. Obviously, the default kernel represents the basic version of the mobile application that has to be offered to the user regardless of the current context.

6.2.2. Variability Generation

At this step, the designer needs to derive the variability of the optional and alternative features modeled in the feature view. Again, based on feature selection, designer will decide on the features to derive as separate components. Three feature groups are concerned with this derivation:

- Optional features in critical feature group
- Optional features in important feature group
- Alternative features in useful feature group.
The derived components represent the variability of the SPL members. They are also associated with context requirements. At runtime, the variability components will be activated on-demand. After assessing the current context, the context decision module will make the decision on which variability components to stop and to activate to adapt to the context change. Contrary to Kernel components, during the execution of the application, it may happen that at a certain point in time, none of the variability components are running.

### 6.2.3. OCL Engine

While "<<Feature dependency>>" in the feature view allows the modeling of the dependencies among features, the rules for deriving the mobile application composite from the feature meta-view are not defined. These rules are needed to govern the feature selection when deriving mobile application SPL members with respect to feature priority, kernel component derivations, variability component derivation, and consistency checking.

To address this issue, derivation rules using Object Constraint Language (OCL) [46] are defined. For instance, each generated kernel component has to include all the critical features. From each critical feature group, exactly one variant of the kernel group is included. Each Kernel component may include an important feature. Optional features in the critical and important features can be mapped to variability component. OCL engine also should verify the feature dependency when cutting the different mobile application member SPL. Table 4 summarizes a set of mapping constraints from the feature view to the Mobile app. Composite view, where components are prepared to derive SPL members.

<table>
<thead>
<tr>
<th>Feature model stereotypes</th>
<th>Mapping Relation with the Mobile App. Composite meta-view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical Feature</td>
<td>1. Each kernel component includes exactly one variant of the kernel group of a critical feature</td>
</tr>
<tr>
<td></td>
<td>2. Optional feature may be mapped to Optional Component in the Composite structure</td>
</tr>
<tr>
<td></td>
<td>3. Each kernel component include ALL the critical features</td>
</tr>
<tr>
<td></td>
<td>4. Each kernel component has NO Optional feature of the critical group.</td>
</tr>
<tr>
<td></td>
<td>5. ContextReq of kernel component is the set of all contextReq of the features it includes</td>
</tr>
<tr>
<td></td>
<td>6. ContextReq of an optional component is the contextReq of its optional feature</td>
</tr>
<tr>
<td>Important Feature</td>
<td>1. Each kernel component may include at most one variant of the kernel group of an important feature</td>
</tr>
<tr>
<td></td>
<td>2. Each kernel component may include some important features</td>
</tr>
<tr>
<td></td>
<td>3. Optional features may be mapped to Optional Component in the composite structure</td>
</tr>
<tr>
<td></td>
<td>4. Each kernel component has NO Optional feature of the important group feature.</td>
</tr>
<tr>
<td>Useful Feature</td>
<td>1. Each Feature alternative is mapped to Optional Component</td>
</tr>
<tr>
<td></td>
<td>2. ContextReq of an Optional component is the ContextReq of the feature it includes</td>
</tr>
<tr>
<td></td>
<td>3. Each kernel component has NO Useful feature.</td>
</tr>
</tbody>
</table>

### 6.2.4. Context Composition Engine

Since Kernel components are obtained by composing different critical features, we propose to add a context composition engine that generates from different contexts the composite context to associate with the kernel composite. In this work, we adopt the same context composition presented in [14]. The context requirement is modeled as a set of attributes. Each attribute has a specific type. The type of an attribute has the role to specify the composition semantics of this attribute. The framework the authors in [14] are describing distinguishes four types: set-based type, metrics-based type, unknown type, and notapplicable type. Composition of two context elements of the first type may lead to the intersection, union, and inclusion of the two sets. In other words, the type of the attribute will dictate its composition semantics. For example, if a given feature is designed for IOS system and another feature version is designed for Android system, the context of the composition is simply the empty set as this composite cannot run in Android or in IOS. In this case the operating system attribute should be modeled as Intersection-Based-Set-Element. To the opposite, GPS availability would be modeled as Union-Based-Set elements. In fact, if
a given feature that needs GPS is composed with a feature that does not need GPS, then the composite needs GPS. At the end of the step, a context is generated for each composite kernel.

![Figure 5: Context element Types [14]](image)

6.3. **Runtime**

Each SPL member application is a composition of a kernel bundle and a combination of variability bundles, driven by the developed feature model.

**6.3.1. Context Acquisition**

This module is responsible for collecting the current user/device context. It relies on the handheld device operating system to collect the information needed. As mentioned previously, operating systems for smartphones and tablets offer a set of APIs made available for developers to collect sensed information and make use of it. It includes: user location using the GPS, network type, and the battery level. Once this information is collected, it is communicated to the context decision module to adjust the behavior of the application accordingly as we will show next.

**6.3.2. Context Decision Module**

The context decision module is responsible for deciding, depending on the current context and the feature model of the application, which bundles to activate.

To make the decision, designers feed the context decision module with:

1. A priority schema for the different elements of the context. In some applications, battery may have the highest priority while in others network speed could be the most decisive factor of the context. Changing this priority schema before launching the application gives more flexibility to answer end-users needs and to serve the application domain.

2. The feature model as defined by the modeler as it has all the information about critical features, important features and their kernel parts as well as the different optional features. From the model, the module will know the list of features that should be provided all the time and respect the <<Exactly-one>> constraint.

**6.3.3. Runtime Manager Module**

This module has two roles:

a. Runtime consistency checking with respect to the feature model. As the application is composed of a set of bundles that may be activated and stopped, it is important to make sure that at any time no inconsistency raises from stopping and activating bundles. For instance, this module ensures that two kernel bundles cannot run at the same time, no alternatives of the same feature are running, to cite few. It in fact, this checks the compliance of the running SPL member with feature dependency as defined in the feature view.

b. It also checks uninterruptible vs. interruptible bundles. In other words, the Runtime manager module checks if the bundle can be interrupted or not based on its properties defined at design.
time (from application requirement point of view). For instance, if the task of updating patient report was already launched, then this task will not be interrupted. Deactivation of this bundle will be postponed until the task is successfully completed unless the network is disconnected. For the sake of simplicity, we made all the tasks of our running example interruptible except the patient report updating task – once it starts, its deactivation will be performed only once the update is successfully completed to avoid losing/corrupting patient data.

7. **Proof of Concept**

In this section, we present an implementation of our framework. This implementation includes a modeling tool based on the multiple variability meta-model described in Figure 1 and a framework to deploy at the mobile device side to run context aware and adaptive mobile application SPL. During our experiments, the tests have been conducted using Samsung Galaxy tablet, model number GT-P7500 with Android version 3.1.

7.1. **Design Time**

At design time, we used the developed modeling tool to model each view as illustrated earlier in the meta-model.
The tool in [47] was extended to cater for mobile application. This tool is built on top of Eclipse Modeling Framework (EMF) [28]. EMF supports model driven engineering by providing a set of tools and libraries that facilitate creating a domain specific languages (DSL). In our work, we used ECore, an EMF metamodel, to describe the multiple-view variability models. For model consistency checking, we exploited ECore’s model annotation capability to embed OCL constraints in the meta-model of each view. The implementation of the tool’s GUI is based on Eclipse Rich Client Platform (RCP). The tool has a separate editor for each view within the framework. It allows the designer to model and validate the created models against the embedded OCL rules. Figure 6 (a) shows the feature model editor where the system designer can define the functional requirements of m-Heath system as Critical, Important, and Useful. Moreover, as seen from the figure, the tool allows modeling the Composite Views and mapping each created component with the features defined earlier. By applying the mapping rules illustrated earlier in section 6.2.3 to the m-health system, the Kernel Component “m-Health main SPL component” is referencing the features that represent the kernel instance of each derive version of m-health SPL. Therefore, all Reporting Text feature Group as well as Display Location are mapped to the Kernel Component. From the other side, the composite view encompasses Optional Components such as “Report video,” “Send Report” and “Send Report” to mention few. Each Optional Component is realizing optional features of <<Critical>> and <<Important>> Feature Group, in addition to alternative features of <<Useful>> Feature Group such as “Plan Real-Time Voice”.

In a further step, the same tool can be used to allow system designers to perform feature selection, where a subset of the original feature model is generated as shown in Figure 6 (b). Each generated feature model of the m-Heath SPL member application should include a “Patient Report” feature where one of the “Reporting Text” mechanism need to be selected. In our example, it can be “Send Critical Data” feature, which is designed as a default feature. In case the designer didn’t perform any selection of one of the features under <<Exactly-One>> feature group, the system will automatically include the default feature. In addition, the designer might choose to include “Identify Nearest Emergency” feature as well. However, based on the defined OCL constrains, the tool allows first to include the kernel part <<Exactly-One>> feature group of <<Important>> feature. That is, the designer has to specify which “Display Location” functionality needs to be operated such as “Display Map” otherwise the default feature will be included. The designer can choose later from the optional features such as “Report Video”, “Send Report” beside the alternative features of “Plan Intervention” <<Useful>> group as “Real Time Voice” feature.

The newly generated Composite View Model of the created m-Health member application contains the only components that are mapped to the selected features. The whole “Composite” of the Kernel and Optional Components form a member of m-Health SPL.

7.2. Run time

Before presenting our mobile application, we first present the result of some tests conducted to measure the battery consumption of OSGI [14] bundles activation in the handheld device. The tests measure the impact on the device performance when dynamically activating and stopping OSGI bundles.

7.2.1. OSGI Bundles deployment in Android platform: Performance Measurement

As shown in Figure 7, the activity of stopping and activating bundles had no significant impact on the battery, which is important for our framework as battery consumption is one of the critical context elements that affects the decision.
In addition, Figure 7 (b) shows that the time needed to activate and stop bundles is in the range of milliseconds and it grows linearly with the number of bundles. The more bundles the OSGI platforms stop and activate at once, the more time needed to achieve this task. Having said that, activating and deactivating 20 bundles at a time will need an average of 1.5 ms, which is not perceived by the user according to our experiments. Our preliminary tests proved that running the OSGI platform on the tablet mentioned previously did not affect its general performance. Further tests are conducted as shown next to confirm this finding.

2) m-Health Client Android Application

Using our framework, we developed an m-Health client Android application that can be used by doctors on the move. Using the application, the doctor can view, and upload patients’ data to the server. We implemented the server side of the application using PHP XAMPP Server. The client application provides ten features illustrated in Table 1. Each feature consists of one or more bundles depending on its type (Critical, Important, or Useful feature). Every bundle is associated with a specific context. If the current context does not satisfy the bundle’s context, the bundle is stopped, and the feature is disabled from the user by shading its option at the graphical user interface level.

The application architecture consists of the main activity (MainActivity) component, which is invoked when the application starts. MainActivity instantiates Apache Felix OSGI framework, where OSGI bundles are loaded. Then, it displays the ten features the application has on the device GUI. Other activities take care of displaying the sub-features such as kernel and optional features in the critical features. These activities are ViewPatientOptionsActivity, KeywordSearchActivity,
ByDateSearchActivity, and ReportActivity. The service is available if the bundle is in an ACTIVE state. On the click of any feature, the bundle service that corresponds to that feature is invoked only if that service is available. In addition to the main thread, BundleManager thread is running in the background, managing the state of the different bundles. This management is achieved by executing a method that keeps running for the lifetime of the application. The functionality of this method is to update the state of the bundles every specific period of time based on contextual criteria. BundleManager acquires the different context parameters with help of the BatteryMonitor, Connectivity, and GPSMonitor classes. These classes provide the battery percentage, the network type and speed, and the GPS status correspondingly. Figure 8 illustrates the class diagram of our Android application.

3) Bundle Manager Algorithm

As mentioned earlier, a BundleManager thread is used to manage the bundle states depending on contextual criteria. We represent the bundle management algorithm by a procedure called MANAGE_BUNDLE_ON_RUN. The functionality of the procedure is to update the state of the different bundles periodically. The algorithm checks the kernel bundles and the optional bundles, and updates their states if needed. Before explaining the algorithm, we first define the notion of context inclusion.

**Definition 1: Context Inclusion**

Let a and b be two contexts. We say that context a includes context b if for a given context c, if c satisfies a then c satisfies b.

Kernel bundles are sorted based on the adaptation goal. In our case, the goal is to provide the end-user with the maximum number of features allowed by the current context. Thus, a kernel bundle A has a higher priority than a kernel bundle B only if A provides more services than B. In case A and B encapsulate both the same features, then A will have higher priority if its context includes the context of B. Therefore, if the context of a kernel bundle is satisfied, then the contexts of all the lower priority kernel bundles are also satisfied. Similarly, if the context of a kernel bundle is not satisfied, then the contexts of all the higher priority kernel bundles are not satisfied. Obviously, the default bundle, implementing the default kernel, has the lowest priority. For a group of kernel bundles, the algorithm ensures that the bundle with the best quality of service (highest priority), and whose context is satisfied will only be running at a certain point in time. In the worst case, the default bundle will be running based on the assumption that some core functionalities of the application should be offered to the user. Additionally, any optional bundle will run, as long as its context is met. Otherwise, the bundle is stopped. The MANAGE_BUNDLE_ON_RUN procedure pseudo-code is illustrated in Figure 9.

The algorithm works as the following. For every feature, the algorithm iterates through its kernel set of bundles, in the descending order of their priority (the higher priority bundle is checked before the lower priority bundle). If a bundle is at the STOPPED state, while its context is satisfied, the algorithm starts that bundle after stopping the lower priority bundle (if one is running). This condition is illustrated in figure 9, lines 4 through 12. On the other hand, if the bundle is at the STARTED state (running), while its context is no longer satisfied, the algorithm stops that bundle and starts the next bundle with lower priority and whose context is satisfied. This condition is illustrated in Figure 9, lines 13 through 27. Note that after stopping the bundle in this condition, the algorithm does not check the context of the higher priority bundles, based on the assumption that their contexts are not satisfied, as mentioned earlier.

Since we aim to keep the highest priority bundle running, the algorithm breaks the iteration once it starts a bundle, or if it reached a running bundle whose context is satisfied (lines 10 and 21). The second iteration of the algorithm is performed through the optional bundles of the feature. This iteration is much simpler. A stopped bundle is started if its context is satisfied, while a running bundle is stopped if its context is no longer satisfied. This iteration is illustrated in Figure 9, lines 29 through 36. The algorithm repeats these two iterations for every feature, and waits for a specific period of time, before running the whole operation once again.

4) Uninterruptible and Interruptible Bundles

Stopping a bundle introduces an issue when a running thread is already launched by this bundle. It raises a question whether we should leave this thread running or kill it immediately when the bundle is stopped. The answer to this question depends on the kind of the process running. Processes such as updating the
patient file should not be killed in order to keep the database organized, and not to leave any mess in the patient file. We refer to bundles with such uninterruptible processes as uninterruptible bundles. Since we are using OSGI framework, we need not to worry about stopping these bundles. This is because the default behavior of OSGI framework does not include killing any bundle process when the bundle is stopped. Instead, the framework gives you the freedom to implement the action to be made once the bundle is stopped. In java, this can be achieved by implementing the BundleActivator interface, and taking the needed action in its stop method. In case the bundle is uninterruptible, we only unregister the bundle service, so it cannot be used for future requests. Figure 1 illustrates our java implementation for the stop method in an uninterruptible bundle.

On the other hand, some processes are interruptible since leaving them running would highly affect our adaptability purpose, and terminating them would not affect the usability of the application. For example, a bundle that contacts the server to download a video file is expected to consume a lot of battery power. If this bundle was running, and the battery percentage dropped to a level that does not satisfy the bundle’s context, we really need to kill the downloading process when we stop the bundle. Otherwise, the adaptability will fail, and the battery will be highly affected.
We refer to bundles with such interruptible processes as interruptible bundles.

Figure 9: MANAGE_BUNDLE_ON_RUN Procedure Pseudo-code

```java
MANAGE_BUNDLE_ON_RUN ( FEATURE_SET )
1. while ( application_is_running )
2.   for each feature in FEATURE_SET
3.     for each bundle in feature KERNEL_BUNDLE_SET SORTED_BY_PRIORITY DESCENDING
4.       if ( bundle.state == STOPPED && current_context SATISFIES bundle.context )
5.         ATOMIC
6.           if ( feature.IS_LOWER_PRIORITY_KERNEL_RUNNING ( bundle ) )
7.             feature.STOP_LOWER_PRIORITY_KERNEL ( bundle )
8.           end if
9.          bundle.START()
10.         break
11.        END ATOMIC
12.       end if
13.     else if ( bundle.state == STARTED )
14.        ATOMIC
15.        bundle.STOP()
16.        lower_priority_kernel_set $< feature.GET_LOWER_PRIORITY_KERNELS ( bundle )
17.        for each l_bundle in lower_priority_kernel_set SORTED_BY_PRIORITY DESCENDING
18.          if ( current_context SATISFIES l_bundle.context )
19.            l_bundle.START()
20.          break
21.        end if
22.      end for
23.    end if
24.  END ATOMIC
25. end if
26. break
27. end if
28. end for
29. for each bundle in feature OPTIONAL_BUNDLE_SET
30.   if ( bundle.state == STOPPED && current_context SATISFIES bundle.context )
31.     bundle.START()
32.   end if
33.   if ( bundle.state == STARTED && current_context DOES NOT SATISFY bundle.context )
34.     bundle.STOP()
35.   end if
36. end for
37. SLEEP ( some_period )
38. end while
```
Unfortunately, Java offers no way to kill a running process. However, it allows the thread to be terminated cooperatively. This means letting the thread to stop itself by sending it a signal or changing the value of a shared variable. In our application, we use a Boolean variable whose value is set to false in the stop method, alerting the thread to terminate. The thread can check the variable value in different ways. For example, if the thread is continuously running, it can check the variable value in a while condition. Another example is checking the variable value during the execution of a sequence of instructions. Figure 10 and Figure 11 illustrate a java code snippet for the stop method in the case of an uninterruptible bundle and an interruptible bundle respectively. Figure 12 illustrates two examples for checking the shared variable value by the bundle thread.

```java
public void stop(BundleContext bundleContext) {
    //unregister bundle service
    context.ungetService(serviceReference);
    //leaving any thread running
}
```

Figure 10 : Uninterruptible Bundle Stop Method

```java
public void stop(BundleContext bundleContext) {
    //unregister bundle service
    context.ungetService(serviceReference);
    //terminate the running thread cooperatively execute =false;
}
```

Figure 11: Interruptible Bundle Stop Method

```java
public void run()
{
    while(execute)
    {
        //do something
    }
}
```

Figure 12: Shared Variable checking

7.2.2. Context Aware Application Testing

We conducted a test to show the battery saving provided by our framework adaptability. The test was done by running the m-Health client application twice under the same conditions. In the first time, we kept the full set of features available to the user, without any context adaptability as it is the case in current mobile applications. In the second test, we ran the application with the adaptability framework. In each of the two tests, we attempted to use the ten features of the application (all features). If the feature service is available (depending on the bundle’s state), we use the feature. Otherwise we move to the next feature, and so on. We repeated the test several times under different contexts (for example, starting at different battery level each time). Results in Figure 13 show that our proposed adaptability framework extends the lifetime of the battery by 30%. In other words, the doctor will have a considerable time saving when using the adaptive application we implemented. We draw the attention that until 70% of battery, the adaptive and non-adaptive applications will have the same battery consumption. However things will change when the battery reaches 70% as the adaptive application will move to a configuration biased towards energy saving.
Figure 14 illustrates the availability of each feature during the testing of the adaptive mobile application. As expected, critical features were available all the time. However, due to context changes, only kernel parts of the features were running 15% of the time. For instance, the full version of F1 was offered to the user only 50% of the time. It is mainly due to the battery drainage in this case. Important features were available only 70% of the time (with a full version running only 50% of the time) and finally useful features were offered only 47% of the time.

8. Conclusion

In this paper, a framework for self-adaptive context-aware mobile application development has been presented. This framework defines a systematic approach to model dynamic adaptation of mobile applications behavior at runtime with respect to context changes. Using SPL concepts, it offers feature priority based dynamic adaptability.
Our framework has the advantages of making use of the established SPL Feature Modeling to manage variability in mobile computing. It provides a meta-model for context aware and self-adaptive mobile applications. It is also based on a multiple-view meta-model for describing feature selection based software product lines. In addition, the treatment of adaptive mobile applications variability concerns is performed in a unified, systematic and platform-independent manner.

We proved in this work that the specification of context requirements in addition to feature modeling can be used to efficiently improve the runtime feature selection. In the future, we intend to further explore the algorithms in the Context Decision Module to improve the selection of variability at runtime.

While the focus of this paper was the definition of the framework and handling runtime variability at the device end, future work will focus on handling variability on the server end by allowing the evolution of the mobile application itself. An interesting direction that we are planning to explore is the evolution of the requirement model at runtime. So far, the runtime adaptation we are suggesting is based on the representation of the application feature model to make adaptation decisions. Unpredictable contexts that have not been taken into consideration while generating the composite components are so far treated in a static manner by deploying default composites selected by the modeler at design time. The research challenge here is to reflect this need by evolution of the feature model itself and re-evaluation of the design-time decision.

REFERENCES


46. OCL 2014; [http://www.omg.org/spec/OCL/].