A Framework for Mobile Application Design

Abdesselam Redouane

Department of Computer Science and Engineering, College of Engineering and Computing
Al Ghurair University, Dubai, UAE

Abstract
Selecting the right components to design a mobile application involve some deep thoughts and difficult decisions to make. In this paper, we present a framework to ease the decision making process. The framework is based on Commercial Off-The-Shelf (COTS) paradigm. COTS techniques aim to reduce development time and hence decrease cost compared to a traditional system development. First, an identification of components from the application requirements is made. Then, for each component, we specify a formal model, which is called the ideal-component. A structured first order predicate calculus is used as a tool to formalize application requirements and obtain these formal models. The evaluation of a possible–component, from a vendor, begins with understanding the features and then an acceptance indicator is calculated. The acceptance equation combines three key factors: requirements and features match, vendor-viability and maintainability. Maintainability is a costly phase in any software system and this framework caters for this issue during the evaluation process. The framework is being investigated with successful results.
Keywords: *Mobile Application, COTS, Predicate Calculus.*

1. Introduction

The ubiquitous of mobile devices has sparked wide spread development of mobile applications. Mobile applications range from game applications, maps, news and social networking, to sophisticated business transactions. High end mobile devices sales are expected to reach high volume of sales all capable of running some applications. New features, e.g. sensors, present new challenges to developers that are not found in traditional software development. The need for a self-adaptive application and hence expressing the requirements formally is very important in mobile applications software engineering [1].

According to the survey in [2] on mobile application development, these applications are not large, only several thousand of lines code and no rigorous process followed to develop these mobile applications. It has been found that many mobile applications exhibit many errors [3], [4], [5].

Designing these applications for an array of devices is not an easy task and rigorous requirements capture for these applications is of paramount importance if these applications are going to be reliable and trustworthy. The same device may change characteristics within few months. Keeping with these changes is a tremendous job for applications upgrade or the application becomes obsolete for future devices.

Recent effort in HTML 5 and PhoneGap allow the development across multi platforms: iOS, Android, window 7 etc … is an attempt to reduce the development effort. But with these technologies there is no access to the native API which hinders the application from accessing the full capabilities of the device.

The decision to select a component, among many, that fit a design is very hard to make. To get it right, a profound thought about the component must be made. The mind of the designer has to be very clear and correct boundaries should be drawn for this component. It is not simple to have such clear and no confusion situation when we have many off the shelf components which might fit the purpose. In this paper we provide a framework that helps ease the decision process.

Commercial off-the-shelf (COTS) techniques aim to reduce development time and hence decrease cost compared to a traditional system development. COTS paradigm has gained considerable attention in the last years among researchers within the software engineering community [6], [7], [8]. It is seen as an alternative effective solution to the conventional costly method of software system development. To lessen the decision making process, we advocate the use of Commercial Off-The-Shelf (COTS) components during a mobile application design. It will bring all the advantages of cost reduction and a minimum time to market a product. Indeed a mobile application designer can concentrate on the application architecture, select their required COTS components from the market, do the integration and perform testing. The approach is highly feasible given the fact that, nowadays, we find more and more ready made components.

In this paper, we propose a framework for mobile application design. The framework is based on COTS techniques and is formal in the sense that we adopt a rigorous requirements specification by the use of a structured first order predicate calculus. We believe that in order to get correct requirements, a formal technique has to be in place. The component requirements is formalized in order to get a deep understanding of what is intended from the component and be able to make an informed decision during the evaluation of a vendor component. It is only through formal techniques that we can achieve such an insight and understanding.

The remainder of this sequel is structured as follows. In section 2, we describe the framework in detail while in section 3 we discuss related work. Section 4 describes a case study and section 5 concludes the paper.

2. Framework Description

The aim of the framework is to help the designer makes the right decision when choosing among many similar components. The framework is based on the following steps:

- Mobile application requirements collection
- Component requirements formalisation
- Acceptance indicator calculation
- Component selection

In the following sub sections these steps are discussed.

2.1 Mobile application requirements collection

The first phase in this framework is a traditional collection of the application requirements. The source of the requirements can be a customer or a manager. A requirements document is written using an informal language where both parties' designers and customers/managers could agree. This informal requirements document should contain sufficient requirements in the sense that a designer can move to the next phase of the system development. The requirements are, then, divided into components and a list of the system components is made.

At this early stage, it is possible that there are some requirements, which are incomplete. The designer, however, has some ideas about these incomplete requirements. In these cases, one should continue with the collected complete requirements and move on to the next phase. The incomplete requirements should be recorded and their places in the system must be clear.

2.2 Component requirements formalization

For each component identified, a formalization of its requirements has to be developed. We use a structured first order predicate calculus to capture the component requirements. It is a very expressive formalism and simple to use. Another advantage is its wide familiarity among designers and it will not be a major obstacle to companies adopting this approach. The specification should state what are the functionalities and properties intended. This requirements specification forms an ideal model and we call it the *ideal-component*. An *ideal-component* captures exactly what is required. One can validate this by passing the specification between developers to verify that the specification is neither missing requirements nor including extra requirements. This specification is the target, which a designer strives to get a component from the market that fits it. In fact, an *ideal-component* is the set of formal requirements for a given system component.

First order predicate calculus is the vehicle being used to formalise the requirements.. A light structuring for this calculus will make the end specification more amenable to analysis. The form which is followed to structure the specification of a requirement, called a definition, is as follows:

definiendum Δ | context |
refinement

where:

- definiendum is the name of the requirement being specified.
- context is a name of a defined requirement.
- Refinement is where the specification of the requirement takes place.

A definition is a logical rule in which a definition is true if and only if the context and the refinement are true:

$$\text{definiendum} \Leftrightarrow \text{context} \wedge \text{refinement}$$

Through chaining via context we can have what is called strong relationship between requirements. This allows easy tracing of requirements. Analysis of specification can be made through the use of the following domain theorem:

$$\text{definiendum} <> \text{context} <> \text{mot definiendum}$$

<> is an exclusive-or operator, in that, this expression states that only one term can be true. The mot definiendum is true when the context is true and the refinement is false. This allows the specifier to negate a logical complex expression without a cumbersome manipulation of variables. This can be summarised in the following rule:

$$\text{mot definiendum} \leftrightarrow \text{defineniddum} \wedge \sim \text{refinement}$$

Once the specification has been developed for a particular component, a search in the market for vendors has to be carried out. The following diagram illustrates this phase:

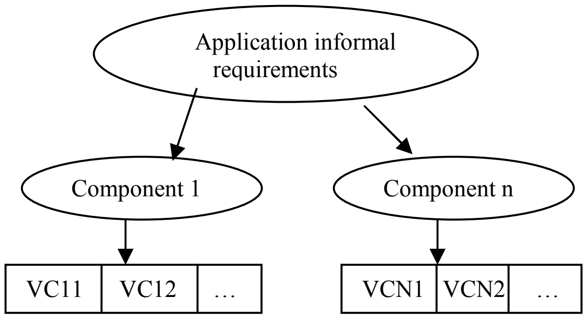

Fig.1. Initial phase

Where: VC11, VC12 are vendor components for Component 1. VCN1, VCN2 are vendor components for Component N. Each component is a possible candidate for the system and we call it a *possible-component*. A *possible-component* is a component that, initially, from the vendor description, we believe that it may fulfil some of the required features of the *ideal–component*. At this stage, we rely on the description given by the vendor for a *possible-component*. Of course, later on we perform the testing of this component to see if it really meets the description given by the vendor.

However, if there are no vendors available a contractor has to be found or an in-house development of the component has to be carried out. This development by a contractor or in-house must comply with the developed specification for this component.

2.3 Acceptance indicator calculation

It is in this sub-section that the framework offer guidance to make the right decision when we have multi components to choose from. In order to rank the *possible-component* candidates, an acceptance indicator needs to be calculated. This indicator is evaluated by the following equation:

Acceptance Indicator % =

$$(\alpha.\text{match} + \beta.\text{vendor-viability} + \delta.\text{maintainability}) \times 5 / (\alpha + \beta + \delta)$$

The above equation combines three important factors: requirements and features match, vendor-viability and component maintainability. They range from 0 to 20. In the following sub-sections we describe these factors in details.

2.3.1 Match Criteria

A comparison has to be made between the ideal-component and a possible-component. That is, between the requirements specification and the features of the vendor component. This comparison has to be made as follows: for each requirement specification from the ideal-component, a feature(s) from the possible-component has to be found. It is understood that the evaluator must be competent with the requirements specification and able to dig deep into the vendor features to see if they match. The match factor will be ranked high if all the requirements are found within the features. It will receive a low score if there are only a few requirements which match. In fact, three scenarios are possible. The first is when a complete match has been found between the ideal-component requirements and vendor features. The second scenario is when the vendor features are less than the ideal-component requirements. In this case, the vendor component is rejected. It is kept, if and only if, there are no other components to choose from. In this case, a trade off has to be made between accepting this component and adding the missing requirements if the source code is available, or developing an in-house component. The final scenario is when the vendor features exceed the ideal-component requirements. In this case, further investigation is required and should answer the following questions:

- Will the extra features add further enhancement to the system?
- Will it harm or degrade the system performance?
- Is it possible to disable these extra features and enable them when they are needed?

2.3.2 Vendor-viability

This factor deals with the vendor viability. Is the vendor reputable and well known in the field? There are two main issues to be concerned with: product quality and customer service. The more qualities we have about a vendor, the greater the mark is for this factor (vendor-viability).
- *Product quality:* In general, the question made: are the products from this vendor reputably known for their qualities? If so, it may be the case that the component at hand will, also, exhibit high qualities. This will give an early impression about the component. However, quality check is an ongoing process and it is only time that can confirm this impression towards the component.
- *Customer service:* It is from the customer service that we get missing information and detailed explanation of the component at hand. If this service is not excellent, then, it will hamper our component understanding and, therefore, may be a cause of cost increase. Again, this service can be, initially, judged on the vendor reputation or through early contacts to see if they react promptly to a request made. One should also seek how reliable all the contact channels are: phone, fax and email.

2.3.3 Maintainability

One of the difficult and delicate issues facing designers is maintainability of vendor components [9]. It is a challenge to the system designer. The main concern is: is the component maintainable? That is, is it possible to add new features by the system developers? Or is it difficult/impossible to do so? One has to weight the vendor service and in-house maintainability. Other related questions are: will you add the fixes or new enhancements the vendor has made? Will you update to new releases from the vendor? What effects of integrating the new release into the already functioning system? What would be an ideal situation is the possibility of maintaining the component in-house. If this is not possible because of lack of resources or the source code is not available the vendor will do the enhancement to the component according to the system designer wishes and not to the vendor wishes. From the answers to these queries and the needs from the system, a mark has to be given to the maintainability factor.

2.3.4 Acceptance indicator coefficients

The match, vendor-viability and maintainability criteria range from 0 to 20. Coefficient values depend on the project and which factor may be carrying more weight than the others. The values should reflect the steady degree of importance between the criteria affecting the evaluation process. Usually, the match criteria are very important and it should carry a high weight. Vendor-viability is neither as important as the match criteria nor as the maintainability criteria and hence it should carry a lesser weight than the match and maintainability. Of course, these weights are adjustable to the designer view of the project.

2.4 Component selection

A *possible-component* from the list with the highest acceptance indicator is picked as the new component for the application.

3. Related Work

In the literature, there are a number of evaluation and selection methods, and in this section we discuss the most related to the presented framework.

COTS-based Integrated System Development (CISD) [10] consists of three phases: identification, evaluation and integration. Identification includes collecting and understanding system requirements and identifying and classifying COTS software products. This stage is similar to our first phase of system requirements collection and division of these requirements into components. In CISD, these components are called service domains. Evaluation consists of the development of prototypes to support further investigation of the candidates of COTS products. Our framewrok does not rely on extensive prototyping. We believe that testing should be a confirmation to a specification and not an exhaustive procedure to component comprehension. In CISD, adapters are developed during the integration phase to interconnect the selected COTS components.

Off-The-Shelf-Option (OTSO) [11] is a method where there exist three phases: search phase, screening and evaluation phase and an analysis phase. COTS candidates are identified in the search phase similar to our method first phase. Evaluation is performed against a set of criteria which are taken from requirements specification, high level design specification, etc…The requirements specification carried out in this method is similar to our framework system component requirements specification. The final selection, in this method, is made in the analysis phase of the evaluation results. The PORE method [12], [13], [14], [15], is a method for selecting software packages. Requirements are considered very important as in the presented framework. For that, PORE uses requirement engineering techniques. Unlike PORE, our framework does not rely on the vendor response to a questionnaire on compliance with system component requirements. In our framework, it is the designer who determines this compliance once a specification, of the system component requirements, has been made. In addition, other techniques are also used such as knowledge engineering, multi-criteria decision-making as in [16], and features analysis to guide the selection of COTS packages.

A related method to PORE is the proactive evaluation method (PE) [17], which allows requirements refinement and redefinition. It uses PORE evaluation templates and context evaluation through prototyping.

The Base Application Software Integration System (BASIS) method [18] is based on synthesizing product evaluation, emerging practices in integration technologies and business priorities. BASIS includes three steps. A component evaluation process, a vendor viability process and a difficulty of integration index calculation. Our framework, also, caters for vendor-viability in the acceptance equation. However, we add another important factor to the equation, that is, maintainability of the evaluated component. The final decision, in BASIS, is based on a single prioritised index of suitability and is called the BASIS indicator.

4. Case Study

In this section, we report on an application where the framework has been applied. The application is wireless and it involves a handheld device that is a Palm OS equipped with a WML browser to extract information via a WAP gateway from some web servers. The application architecture is depicted in Figure 2:

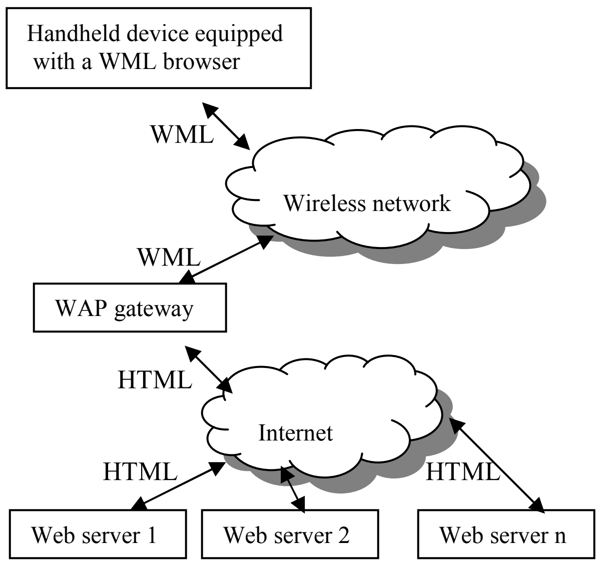

Fig. 2. Wireless application architecture

The requirement is that we would like to add a security layer to the application in order for the user to send and receive data securely. At this early stage, we have identified the following requirements for the new component:

- Data should be correct and has not been corrupted during the transfer in either direction.
- User identity is not revealed to the servers.
- Authentication – the client can authenticate the server where the required data reside, and the server should, also, be able to authenticate the client.

4.1 Specifying the requirements

The *ideal-component* requirements can be specified, in a structured first order predicate calculus, as follows:

Transmit data from a to b requirement specification:
transmit(d, a, b) Δ | application |
know(a, d) → know(b, d)
Requesting data requirement specification:
want(a, d, b) Δ | application |
~ know(a, d) ^ know(b, d)

Data correctness requirement specification:
data_correctness Δ | application |
∀s, cl, d ((server(s) ^ client(cl) ^ data(d) ^
tramsmit(d,cl,s)) → correct(d))

User identity requirement specification:
userID Δ | application |
∀s, cl, d (server(s) ^ client(cl) ^ data(d) ^
tramsmit(d,cl,s) → ~ (know(s,client-id)))

Authentication requirement specification:
authentication Δ | application |
∀s, cl, d ((server(s) ^ client(cl) ^ data(d) ^ want(cl,d,s))
→ (authenticate(s,cl) ^ authenticate(cl,s))

In the above specifications, we have stated formally what the *ideal-component* must satisfy. The predicates server(s), client(cl) and correct(d) are self explanatory. The definition transmit(d,a,b) has the meaning of the data can be sent in both directions from a to b and vice versa that is client and server. The definition want(a,d,b) has the meaning of a wants the data (d) from b. The prdicate know(s,client-id) has the meaning of the server knowing the client id.. Finally, the definition authenticate(a,b) has the meaning of (a) performing an authentication on (b). Note that in these definitions the context in which these definition have meaning is the application environment.

4.2 Evaluation of components

In this project the match criteria is very important and we require that the selected component should match the *ideal-component* perfectly. Maintainability is of high importance, as we would like to tailor the component to our needs. Hence, the following values for the coefficients of the acceptance indicator equation have been taken:

$\alpha = 7$	for the match coefficient
$\beta = 3$	for the vendor-viability coefficient
$\delta = 5$	for the maintainability coefficient

After substituting α, β and δ by these values the acceptance indicator equation becomes:

Acceptance Indicator % =
(7x match + 3 x vendor-viability +
5 x maintainability) x 5 / 15

A market search for the components that may fulfil the *ideal-component* specification came up with two components: one from RSA [19] and the other from Certicom [20]. These modules, if they are going to be accepted, their features must satisfy the *ideal-component* requirements specification.

From the RSA module description, features identification and understanding have been carried out. Then, the following marks for match, vendor-viability and maintainability factors, 14, 17, 11, have been made, respectively. Note that these marks are over 20. The acceptance indicator can now be calculated:

Acceptance Indicator % =
(7x14 + 3x17 + 5x11) x 5) / 15 = 68.00

The second component candidate in the list is the Security Builder SSL [15] from Certicom. The component allows mutual authentication between sever and client. It has the added advantage that it is a tool kit where we can choose the options, which we like to put in the security layer. Hence, its flexibility and therefore maintainability is highly feasible. For this reason, this component is impressive to our project. High ranking have been given to this product. These values are: 17, 17 and 18 for the match, vendor-viability and maintainability factors, respectively. The acceptance indicator for this component can now be calculated:

Acceptance Indicator % =

$$(7x17 + 3x17 + 5x18) x 5) / 15 = 84.66$$

The result of the evaluation process is that the Security Builder SSL from Certicom has been selected for this project. The development of the application has indeed confirmed this choice in terms of flexibility of this module.

5. Conclusion

We have presented a framework for mobile application design. The framework is based on COTS paradigm. It uses a structured first order predicate calculus as the main vehicle to specify application component requirements. To help the designer in the evaluation process an acceptance indicator is developed. The acceptance indicator equation is based on three main factors: matching between system component requirements specification and vendor component features, vendor-viability and maintainability.

Structured first order predicate calculus has been adopted to write requirements specification. This will lead to modularisation of the specification. Chaining of requirements specification via context can make the analysis and tracing of the specification at reach. Ambiguities of specifications can be resolved in conjunction with the use of the domain theorem concept.

Component maintainability is very important and the introduction of maintainability in the early stages of the evaluation process is one of the main contributions of the framework. A component could be rejected for lack of flexibilities in enhancing existing features, adding new ones or removing existing features. We believe that the presented framework will be of assistance to software designers in making an informed decision when a choice between similar components has to be made.

REFERENCES

[1] Dehlinger, J., & Dixon, J., "Mobile Application Software Engineering: Challenges and Research Directions", Proc. of the Workshop on Mobile Software Engineering.Springer, 2011, pp. 29– 32

[2] Wasserman, A, "Software Engineering Issues for Mobile Application Developmen", *FoSER'10*, 397 – 400, 2010.

[3] Muccini, H., Francesco, A., & Esposito, P., "Software Testing of Mobile Applications: Challenges and Future Research Directions". 7th IEEE/ACM International Workshop on Automation of Software Test, 2012, 29 – 35.

[4] Amalfitano, D., Fasolino A., & P. Tramontana, "A GUI Crawling-Based Technique for Android Mobile Application Testing". Third International Workshop on TESTing Techniques & Experimentation Benchmarks for Event- Driven Software, 2011, IEEE CS Press, pp. 252- 261.

[5] Amalfitano, D., Fasolino, A., Carmine, S., Tramontana, P., & Memon, A, "Using GUI Ripping for Automated Testing of Android Applications", Proc. 27th IEEE international conference on Automated software engineering, 2012, 258 – 261.

[6] Software Engineering Institute, Carnegie Mellon University: Annotated Bibliography of COTS Software Evaluation. Available at: http://www.sei.cmu.edu/cbs/papers/eval_bib.html (1998, 1999).

[7] J. Dean, P. Oberndorf, M. Vigder (Eds): Proceedings of the 2nd Workshop on COTS Software. Limerick (Ireland), June 2000.

[8] S. Sedigh-Ali, A. Ghafoor, R. Paul: Software Engineering Metrics for COTS-Based Systems. Computer, Vol. 34(5), May (2001) 44 – 50.

[9] V. Tran and D. Liu: A Risk-Mitigating Model for the Development of Reliable and Maintainable Large-Scale Commercial-Off-The-Shelf Integrated Software Systems. In Proceedings of the 1997 Annual Reliability and Maintainability Symposium, Jan (1997) 361 – 367.

[10] C. Abts: COTS-Based Systems (CBS) Functional Density – A Heuristic for Better Design. Lecture Notes in Computer Science, Vol. 2255, Springer-Verlag, (2002) 1 - 9.

[11] J. Kontio: A Case Study in Applying a Systematic Method for COTS Selection. In the 18th International Conference on Software Engineering, (1996) 201 – 209.

[12] N. Maiden, C. Ncube, A. Moore: Lessons Learned During the Requirements Acquisition for COTS Systems. Communications of the ACM, Vol. 40(12), December (1997) 21-25.

[13] N. Maiden, C. Ncube: Acquiring COTS Software Selection Requirements. IEEE Software, Vol. 15(2), March (1998) 46-56.

[14] C. Ncube: A Requirements Engineering Method for COTS-Based System Development. PhD Thesis, Center for Human-Computer Interaction Design, School of Informatics, City University, London, (2000).

[15] N. Maiden, H. Kim, and C. Ncube: Rethinking Process Guidance for Selecting Software Components. Lecture Notes in Computer Science, Vol. 2255, 151 –164.

[16] P.K.Lawlis, K. Mark, D. Thomas, T. Courtheyn: A Formal Process for Evaluating COTS Software Products, Computer, Vol. 34(5), May (2001) 58 – 63.

[17] J. Dean: An Evaluation Method for COTS Software Products. In Proceedings of the Twelfth Annual Software Technology Conference, Salt Lake City, Utah, April 30 - May 5, 2000.

[18] K. Ballunio, B. Scalzo, L. Rose: Risk Reduction in COTS Software Selection with BASIS. Lecture Notes in Computer Science, Vol. 2255, Springer-Verlag, (2002) 31 - 43.

[19] RSA WTLS-C Module:
http://www.rsasecurity.com/products/bsafe/wtlsc.html

[20] Security Builder SSL Certicom:
http://www.certicom.com

Dr. Abdesselam Redouane is currently with the college of engineering and computing, Al Ghurair University, Dubai. He received his PhD in Computer Science from Manchester University, UK. His research interest lies in the area of the application of software engineering techniques to new emerging technologies like web and mobile applications. He is also interested in computer security and especially in access control. He is an associate editor of the International Engineering Letter.