A Driven-Context Composition mechanism for Mobile Applications: Metamodeling Approach

Afrah Djeddar, Hakim Bendjenna, Abdelkrim Amirat, Philippe Roose

A Driven-Context Composition mechanism for Mobile Applications: Metamodeling Approach

Afrah djeddar[a], Hakim Bendjenna[a], Abdelkrim Amirat[b], Philippe Roose[c]

[a]*LAMIS Laboratory, University of Tebessa, Algeria*
[b]*LIM Laboratory, University of Souk Ahras, Algeria*
[c]*LIUPPA Laboratory, University of Pau, France*
{afrah-djeddar@hotmail.fr, hbendjenna@yahoo.fr,
abdelkrim.amirat@yahoo.com, Philippe.Roose@iutbayonne.univ-pau.fr }

Abstract

Nowadays, most human activities rely on the use of mobile devices. The recent massive adoption of this technology explains the increasing demand for specific software. In spite of the large number of available mobile applications (apps) with their different implementation forms, the user's requirements differ from one to another. To tackle this issue and also to cope with the heterogeneous settings offered by the mobile devices available, there is a need for a composition mechanism to take advantage from existing services for developing mobile apps according to user's needs and adaptive to their execution environment. Several composition mechanisms were developed in order to meet the user's requirements using existing software entities. However, these existing approaches do not have a global vision of mobile apps composition (e.g. limiting the composition objects to a one type: components only, services only…etc.). In this paper, we propose a composition mechanism for developing context-aware mobile apps. It enables the composition of existing software entities independently from their implementation platforms (i.e. reusing homogeneous or heterogeneous software entities). It also aims to customize the behavior of the desired mobile app according to the various contextual information of the mobile device. To achieve this goal we follow a metamodeling approach. We propose description languages to define the desired mobile app at several abstract levels and we implement the passage among these descriptions using transformation mechanisms. A case study is presented to illustrate the applicability and the effectiveness of this approach.

Keywords: Composition mechanism, Mobile Apps, Reusing Software Entities, Heterogeneity, Context-aware, Metamodeling.

1. Introduction

The mobile domain has received considerable interest. Increasingly, we use mobile apps in our daily life activities in order to support our needs for information, communication or leisure [1]. For this reason, the usage and development of mobile apps has grown exponentially. According to Gartner, by 2016 more than 300 billion apps will be downloaded yearly [2]. In spite of that, the great demand of new mobile apps remains in relevant growth in order to meet social needs in the daily life of each user.

Software developpement for mobile devices is an emerging research area that faces different challenges due to the heterogeneity features of mobile devices [3]. Mobile platforms are quickly evolving including divers capabilities as available sensors and networking hardware, input modes… etc. [4]. To deal with these heterogeneous characteristics, mobile apps must be as adaptive as possible to their contextual information. Otherwise, runtime conditions of the desired mobile app must be conform to the current context of the mobile

device.

These issues prove the need to a composition mechanism of mobile apps in order to achieve the desired functionalities while considering the different mobile device hardware and software characteristics. To the best of our knowledge, there is no such composition mechanism in the mobile environment which designed to consider at the same time various factors like: user's preferences, software entities kind, mobile devices characteristics, and the current state of the different available resources (e.g. Wi-Fi, GPS, Battery-level).

Thus, by taking advantage from existing works, we present in this paper a new approach for the composition of context-aware mobile apps. By this approach, we try to fulfill a key software engineering challenge: the reusability of existing entities [5] regardless theirs implementation platforms and thus the mobile devices heterogeneity challenge.

Thereby, the main objective of this paper is twofold: First, compose mobile apps using existing software entities regardless their implementation type in order to maximize the benefits from existing functionalities and thus satisfy user's requirements. Second, consider the different contextual information of the mobile device during the composition in order to obtain mobile apps that are adaptive to their run-time environment. Therefore, our approach addresses the heterogeneity problems (i.e. software entities heterogeneity, mobile devices heterogeneity) from the architectural perspective. It allows endogenous composition (i.e. connecting software entities of the same type) or exogenous composition (i.e. connecting software entities of different type) of adaptive mobile apps. The exogenous composition overpasses the limit of the different existing composition approach [6], [7] that is: "the reuse of one kind of software entities to compose an app". Furthermore, we aim to provide mobile apps architectural description which is different from the existing ones (i.e. component based approach, service based approach [8]).

In this research work, we aim to perform the composition task in an intelligent manner: (1) Starting by identifying the needed functionalities ignoring how they will be implemented (i.e. without the limitation of the desired app usability). (2) The choice of concrete software entities that will be used to implement the identified functionalities is performed with respect to the mobile device contextual information such as availability of GPS device, the presence of an auto-focus camera, or battery level. (3) The selected concrete entities will be composed using *mediators* in the case of heterogeneous coordination. On this basis, the same mobile app may be deployed and adapted with any mobile device by recomposing it according to the new contextual information. Otherwise, the possibility to support the same mobile app in a range of heterogeneous mobile devices (e.g. cell phones, smartphones, tablets…etc.).

The reminder of this paper is organized as follows: for a clear and effective explanation of the proposed approach, we present a composition scenario illustrated in Section 2 and used as a reference example throughout the paper. In Section 3, we present the context of our research work. Also, we present an overview of the proposed composition mechanism and we explain its operative mechanism. While Section 4, presents the followed methods and used tools to release this composition mechanism. In Section 5, we give the prototype implementation details for the presented example using our proposed approach. Section 6 analyses some existing composition approaches and evaluates our proposed composition mechanism with them. Finally, Section 7 draws some conclusions and perspectives of this research work.

2. A motivating example: Composition Scenario

In this section, we based on the example of ShopReview App discussed in the work of *Cugola* presented in [4] to explain the motivations behind our research work and the main encountered problems. ShopReview App provides the users with alternatives nearby places where the same product is sold at a more with a convenient price when a user publishes the price of a product she found in a certain shop.

Then, it allows users to share their opinion (i.e. various information concerning the products and the shops) using social network such as Twitter. To perform this app following to the composition mechanism, developer selects firstly the different needed concrete entities (CE). Then, he composes theses selected entities in order to obtain the desired CMA. Following to the selected concrete entities presented in the Table 1, the desired ShopReview mobile app needs: GPS for executing the CE 6 and Wi-Fi for CE 5, CE 7, and CE 8. Also, it needs an Auto-Focus Camera to be able to execute the CE 2.

	Needed Concrete entities
CE 1	Component: photo acquisition from mobile camera.
CE 2	Component: local barcode recognition.
CE 3	Service: translate barcode into product's name.
CE 4	Component: textual input from the user of the product's price.
CE 5	Service: more convenient price.
CE 6	Component: user localization via GPS.
CE 7	Service: other nearby shops which offer the product at a more convenient price.
CE 8	Application: Share information on Twitter.

Table 1. Needed concrete entities to compose ShopReview mobile app

After having completed the composition task we found out that the obtained ShopReview App does not work with the mobile device which its courant context is: Has-GPS is GPS-disabled, Has-Wi-Fi is Wi-Fi-enabled, Has-Fixed-Focus-Camera, Battery Level is 50 %, and Free Storage capacity is: 500 MB. Thus, the choice of CE 2, CE 6 without the respect of the courant context limits the usability of the composed app. CE 2 does not allows correct barcode recognition of an image acquired with fixed-focus camera, and CE 6 cannot be executed while the state of GPS is disabled. Another major challenge is the difficulty to release the collaboration between heterogeneous re-used entities (see Fig 1).

Our approach treats these rising problems during the composition of a mobile app (i.e. the difficulty or impossibility to connect heterogeneous software entities without adaptation and to deploy the composite app in an inappropriate environment).

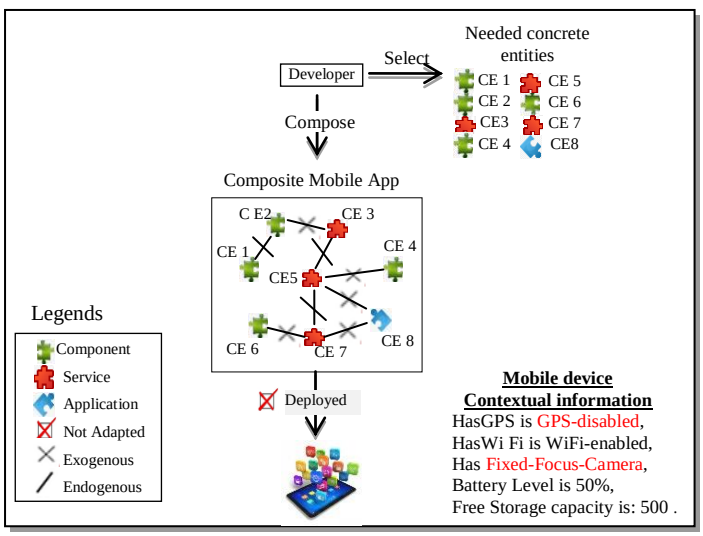

Fig. 1. Composition Scenario for shop Review App

3. Our Approach

3.1. The global context

In order to meet the needs of users by reusing the existing software entities, a lot of composition mechanisms were developed (see Section 6). In this paper, we consider that the composition mechanism can be classified into *Macro* and *Micro* composition. Where the *Macro composition* concerns the composition of classical apps and reflects the composition in the broad sense (i.e. without any constraints). The *Micro composition* concerns the composition of mobile apps which need to be submitted and adapted to several constraints of the execution environment. The notion of *Micro* refers to the obligation of composition with respect to the limited resources offered by the mobile device and its execution state variability.

We remind that the main objective of this research work is the developpement of adaptive mobile apps by composing already existing software entities (i.e. components, services, or apps) according to the user's preferences. The global context of our research work is presented in Fig 2. The composed apps can be built as endogenous or exogenous composition and whose characteristics strongly depend on the software entities which they integrate. This composition mechanism is dedicated to the mobile environment and performed from an architectural point. Furthermore, mobile devices in which the desired apps will be deployed are characterized by various settings. Therefore, the generated mobile apps must be adaptive to their contextual information. For this reason, our composition mechanism takes into account and treats the mobile devices heterogeneity.

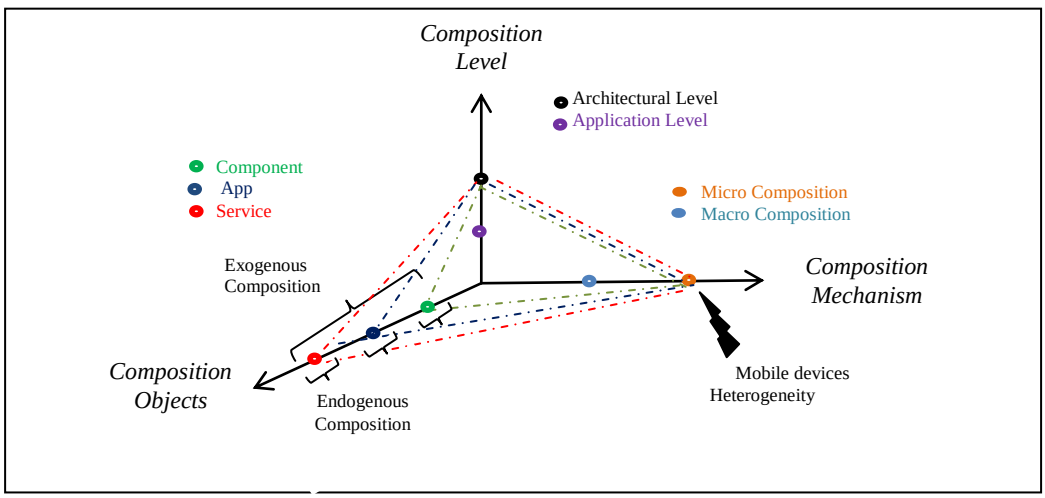

Fig. 2. Context of our research work

3.2. An Overview

Our approach allows the composition of mobile apps by reusing any kind of software entities in order to take advantage of their services regardless to their implementation details. It also aims to respect the limited resources offered by the mobile device in which the composed mobile app will be deployed in order to build adaptive mobile apps.

In order to achieve our objectives we propose to: (1) Elaborate a functional model by identifying all desired functionalities and thus the different composition relations between theses functionalities. (2) Refine the obtained functional model by attaching for any functionality the different information of the best suited concrete software entity that will be used to implement it (i.e. that is adaptive to the mobile device features). (3) Generate the Composite Mobile App (CMA) architectural model by replacing any functionality by its selected concrete software entity (i.e. service, component, or app) including any needed adaptors in order to eliminate heterogeneity problems.

The passage among these different models is represented with transformation mechanisms where: (a) The passage from the functional model to the refined one consists in a refining task and corresponds to an endogenous transformation [9]. (b) The passage from the refining functional model to the CMA architectural model consists in a projection task and corresponds to an exogenous transformation [10].

The Fig 3 represents an overview on the functioning of our composition mechanism which aims to generate the detailed architecture of the desired CMA from a functional representation.

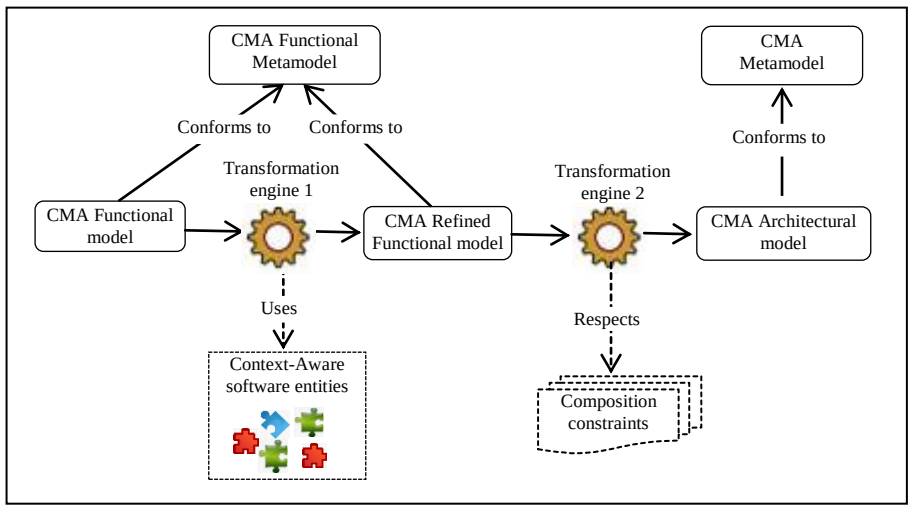

Fig. 3. Overview of the proposed composition mechanism

In the following sub-sections, we introduce our proposed composition mechanism and we explain its operative mechanism to help developing context-aware mobile apps.

3.2.1. Transformation Engine 1: Refining the CMA functional model

The proposed composition mechanism aims, as a first step, to define abstractly the desired mobile app. Thereby, it must firstly focus on the definition of the different functionalities which are required to accomplish the future mobile app's goal and providing the interdependencies between these desired functionalities (i.e. performing a collaboration task while specifies the different invocation orders and exchanged data).

The metamodel presented in the Fig 4 (yellow classes) allows the former of the mobile app to elaborate the functional model. In this way, the CMA functional metamodel represents a high level functional description of the desired mobile app. It allows, in order to obtain abstractly the needed composite mobile app, the defining of primitive or composite functionalities showing the different needed composition relations. A primitive functionality refers to an abstract action; it has an identifier, name, description and a set of input and output data. A composite functionality is a set of primitive functionalities. The composition relations specify *Workflow* and *Dataflow* between identified functionalities. *Workflow* schedules the invocations of functionalities (i.e. establishes precedence links) whereas D*ataflow* expresses the data exchanged, inputs and outputs data, between them (i.e. establishes use links). CMA functional architecture aims to represent the desired mobile app independently of any technology or application domain.

The figure 5.a represents the functional model of the ShopReview app presented in the motivating example. This app needs to execute a set of primitive functionalities in a specific order to achieve its goal. Firstly, we take a picture for the product which we want to buy. Based on the obtained image, it must extract the product's barcode (i.e. Barcode recognition from the taken picture). After having the product's barcode, we need to translate it into string in order to have the product's name. The next functionality aims to collect the product's price from the user. Using the product's name and based-on the given price, other functionality is needed to retrieve, from the internet, the more convenient price offered on e-commerce sites. After, it must retrieve the current user location in order to use the obtained position to retrieve, through the internet, other nearby shops which offer the product at a more convenient price. Finally the user can share the price of the product found in a given shop and also its opinion on twitter.

Developing apps for mobile environment is submitted to several execution conditions. Thus, mobile apps are restrained by the limited resources of the mobile device in which they will be deployed such as: small memory, a battery powered computing environment and availability of some devices (e.g. Wi-Fi, GPS, Auto-

focus camera ...etc.) [11].

An important objective of our approach is to compose mobile apps that are adaptive to their runtime environment. Thereby, in order to elaborate the desired CMA towards a specific mobile device we need to attach any identified functionality with its appropriate context-aware concrete software entity that will be used to implement it (i.e. obtaining a CMA refined functional model which conforms to the refined functional metamodel presented in the Fig. 4 ; see red classes). We associate this ability to the *Transformation Engine* 1. Each concrete entity is represented with its: implementation type (such as: service, component, or app) and its execution constraints (i.e. necessary conditions so that it can function properly). These latter represent the different features required to deploy and execute this concrete entity such as needed capacity storage (size metric), needed energy (consumption energy metric), needed devices (Wi-Fi, GPS, and Camera metrics).

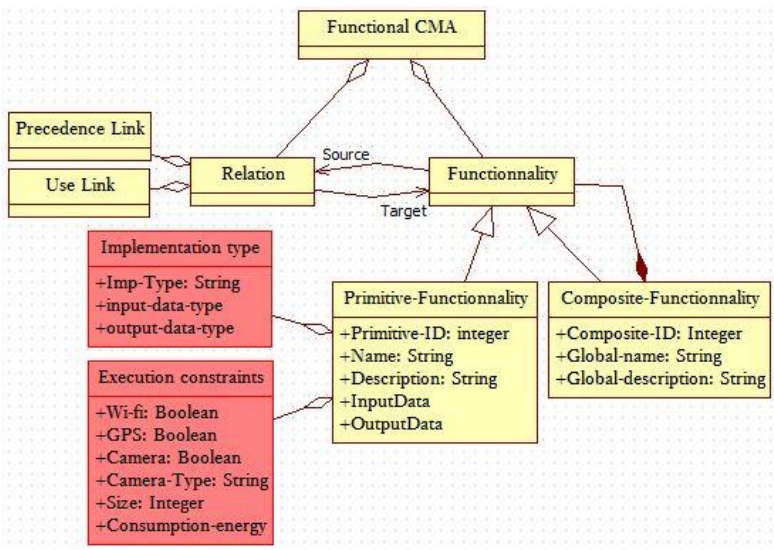

Fig. 4. CMA functional / refined functional metamodel

Transformation engine 1 needs, as source information, a set of suitable concrete entities which much user's requirements described in the functional model and compatible with the device mobile features. These needed context-aware concrete entities (i.e. conforms to the current context of the mobile device) are the result of the selection task of our proposed composition process presented in [12]. If we take for example the functionality Read Barcode, we can find several concrete entities that can be used to implement it where each one of them has its appropriate execution conditions and a specific implementation. The functionality read barcode reflects an abstract software entity which aims to read the barcode that represent an output data of integer type, written in the picture of the product which is an input data of image type. This functionality can be implemented either by a component concrete entity (CE1: local barcode recognition) or a service concrete entity (CE2: remote barcode recognition). These latter can be executed correctly only on mobile devices with an auto-focus camera and does not work properly on other devices. According to the presented motivating example, CE2 is not adaptive to the mobile device context because we assumed that the mobile device in which we will deploy the composite ShopReview app has a fixed-focus camera. Thus, in his case, CE1 is the concrete entity that will be selected to implement the functionality read barcode. It aims to invoke an external service with blurry decoder algorithm to process the acquired image and extract the product's barcode from a picture taken using a fixed-focus camera (see the refined functional model of the ShopReview app illustrated in the Fig. 5.b).

Consequently, *Transformation Engine* 1 corresponds to a refining task providing an adaptive CAM functional model dedicated to be executed in specific mobile device.

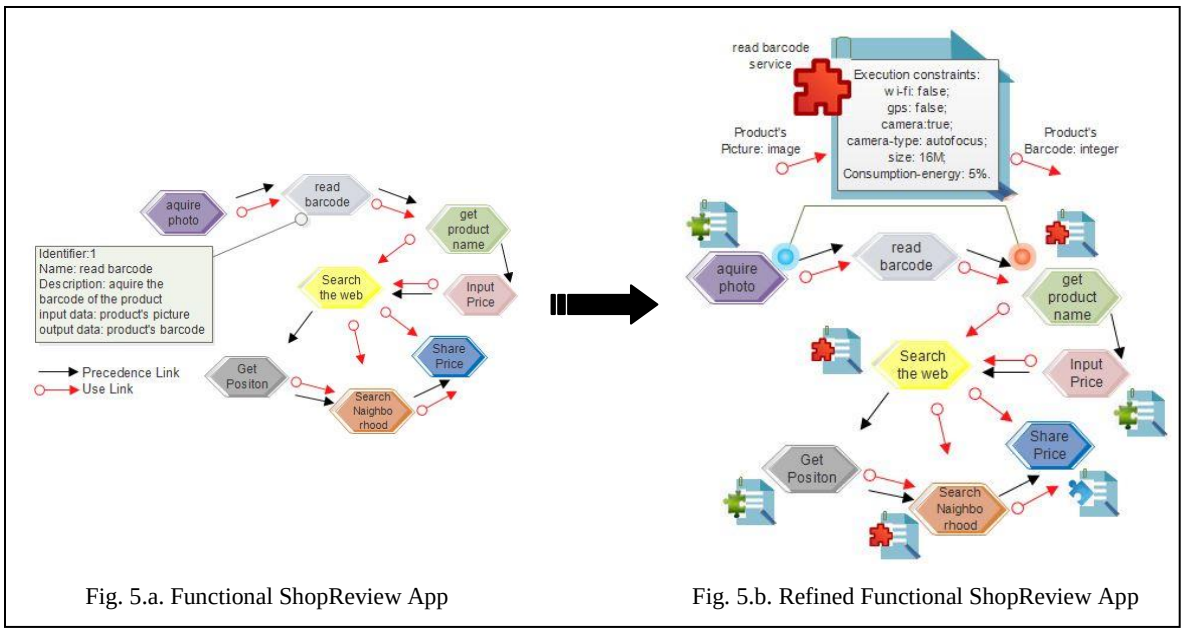

Fig. 5.a. Functional ShopReview App

Fig. 5.b. Refined Functional ShopReview App

Fig. 5. *Transformation engine 1* result

3.2.2. Transformation Engine 2: Generating the CMA architectural model

In our approach, each desired CMA will be developed for a specific mobile environment. So, after having the CMA refined functional model we focus on the detailed description of the desired mobile app. We are based-on the choice of the implementation type of any functionality to construct this detailed architecture. The most important role of the *Transformation Engine 2* is to detect and specify the different heterogeneity points that can arise when composing the selected context-aware concrete entities. This *Transformation Engine* consists to do a projection operation in which any functionality in the CMA refined functional model will be represented with its appropriate context-aware concrete entity. As a result, we will have an architectural model of the desired mobile app which conforms to the CMA architectural metamodel shown in the Fig. 6. This latter represents a high level description of the different context-aware concrete software entities and also the necessary mediators that can be attached on the interdependences between theses re-used concrete entities in order to eliminate the heterogeneity among them. Through this procedure, this architectural description aims to represent the desired CMA with services, components, apps separately or with heterogeneous concrete entities. Therefore, it allows building mobile apps as endogenous or exogenous composition. Each concrete entity has an execution profile that contains all necessary conditions for its execution.

However, *Transformation Engine* 2 must respect several composition constraints to can achieve its goal. Respecting composition constraints corresponds to a mediation task that aims to detect the heterogeneity points and attach the composition relations with the different needed mediators. Heterogeneity issue during the composition task presents two kinds of heterogeneity problems: (a) Heterogeneity nature of entities, (b) Heterogeneity type of entities [12].

The proposed approach aims to solve and overcome these heterogeneity problems by proposing two kinds of mediators: (a) *Endogenous mediators* which are intended to eliminate the heterogeneity between two concrete entities of different natures that cannot directly communicate, (b) *Exogenous mediators* which overcome the heterogeneity between two concrete entities of different type [12].

Therefore, *Transformation Engine 2* has the potential to associate during the composition task the different composition relations with *exogenous mediators* (i.e. encapsulate the source and the target entities with homogenous and well-formed interfaces) and/or *endogenous mediators* (i.e. attaching a mediation services) in

the case of the presence of a heterogeneous and/or exogenous composition; In contrary of homogenous endogenous composition that allows composing software entities with the same type and the same nature by a simple relationship and, therefore, without any adaptation. Then, our approach aims to detect and to eliminate heterogeneity problems from the architectural perspective.

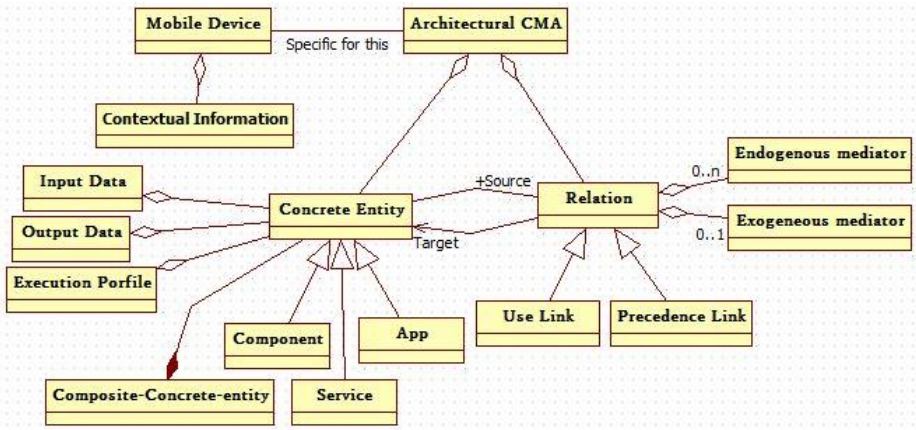

Fig. 6. CMA architectural metamodel

The Fig.7 shows the architectural model of the ShopReview app including any necessary adaptors. Following our example, the functionality read barcode will be replaced with its selected concrete entities CE1 which is of type component. After having replaced each functionality defined on the refined functional model with its appropriate concrete entity, *transformation engine 2* manages the different Use links and Precedence links defined between these functionalities in order to identify if it is necessary to associate these composition relations with mediators in terms of the implementation type of the source entity and the target entity and also the nature of the exchanged data among them (i.e. detecting and managing the heterogeneity points).

If we take for example the CE2 and CE3 which are of heterogeneous type, in this case, the precedence link that connects these two entities must be attached with an exogenous mediator. Thus, the CE1 provides the barcode data of integer type and the CE3 needs to an input data of integer type so that it can carry out its task. Consequently, the exchanged data are not of heterogeneous nature and these tow related entities does not require any data transformation so they could directly communicate. Thus, the use link does not need to be attached with any mediator.

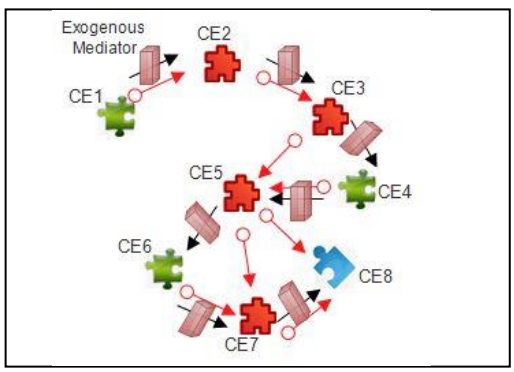

Fig. 7. *Transformation engine 2* result

4. Methods and tools

One of the main goals of our approach is to present specific languages to describe the functional architecture and also the detailed architecture of the desired CMA. To describe the CMA functional model,

subject of the composition task, we define a specific description language in order to represent the desired CMA at abstract level independently from any execution environment. The proposed metamodel of this description language is illustrated in the Fig 8. This latter is defined with EMF technology (Eclipse Modeling Language) under the extension *ecore* [13].

After elaborating the functional architecture using its proposed specific description language and before generating the CMA detailed architecture, we need to refine the functional model by attaching for any defined functionality the different information of the best suited concrete entity that will implement it. The Fig 7 illustrates the necessary architectural elements used to obtain this refined functional model (i.e. red classes).

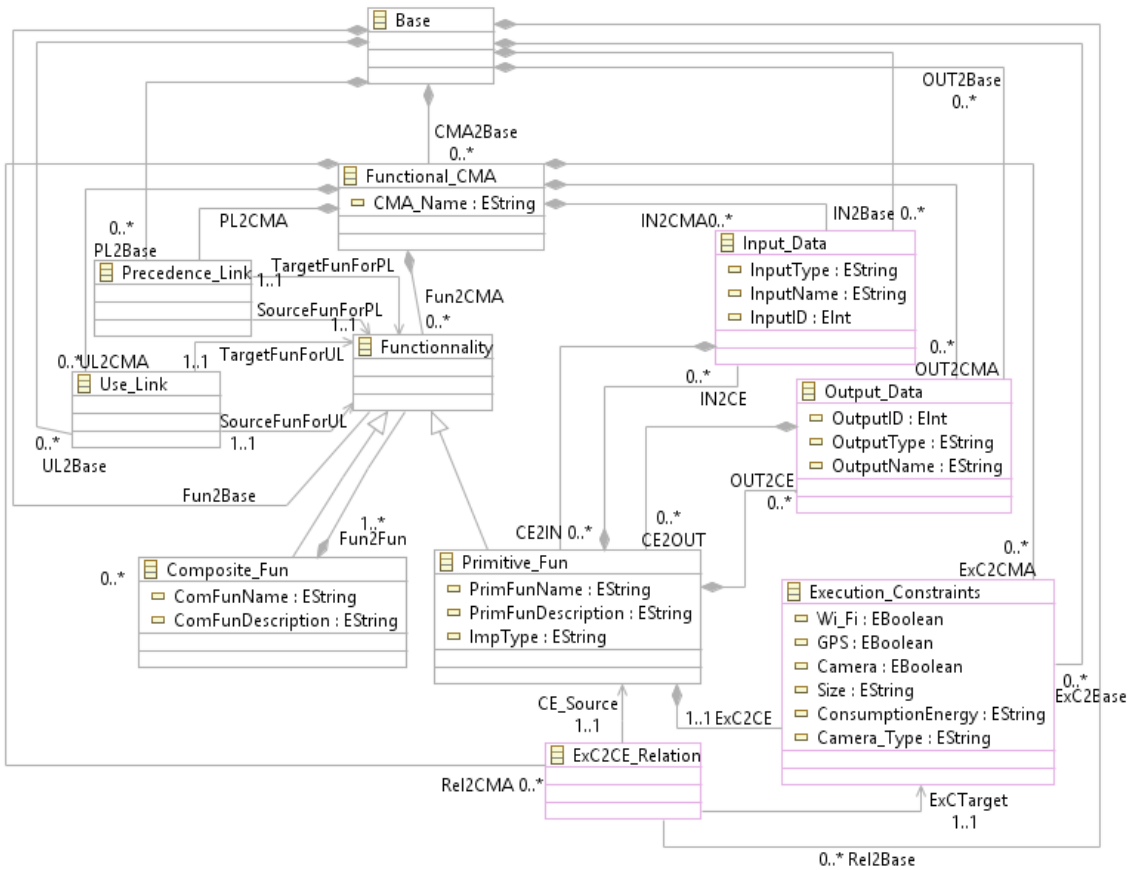

Fig. 8. CMA Functional / Refined Functional Metamodel.ecore

Also, we do need another description language to define the CMA architectural model which is the target of the composition task. The metamodel of this description language is illustrated in the Fig 8. The proposed meta-models are implemented following all steps defined in GMF technology (Graphical Modeling Language) [14] in order to obtain description pallet representing all elements and relations used to describe any functional architecture of any desired CMA, so that we obtain a graphical representation for the CMA detailed architecture.

We implement the passage among these different models using models transformation mechanism [15]. We use ALT language (Atlas Transformation Language) [16] to define the different needed passage rules and java technology to invoke and execute them. Concerning the *transformation engine* 1, we use CMA functional metamodel (expressed in EMF technology) as source and at the same time as target metamodel. It corresponds to endogenous passage as illustrated in the Listing 1.a. However, an architect focuses on the definition of the functionalities which are required in his desired CMA and provides the interdependencies

between these functionalities. For such a purpose, he defines the CMA functional model in a graphical format using the proposed specific pallet.

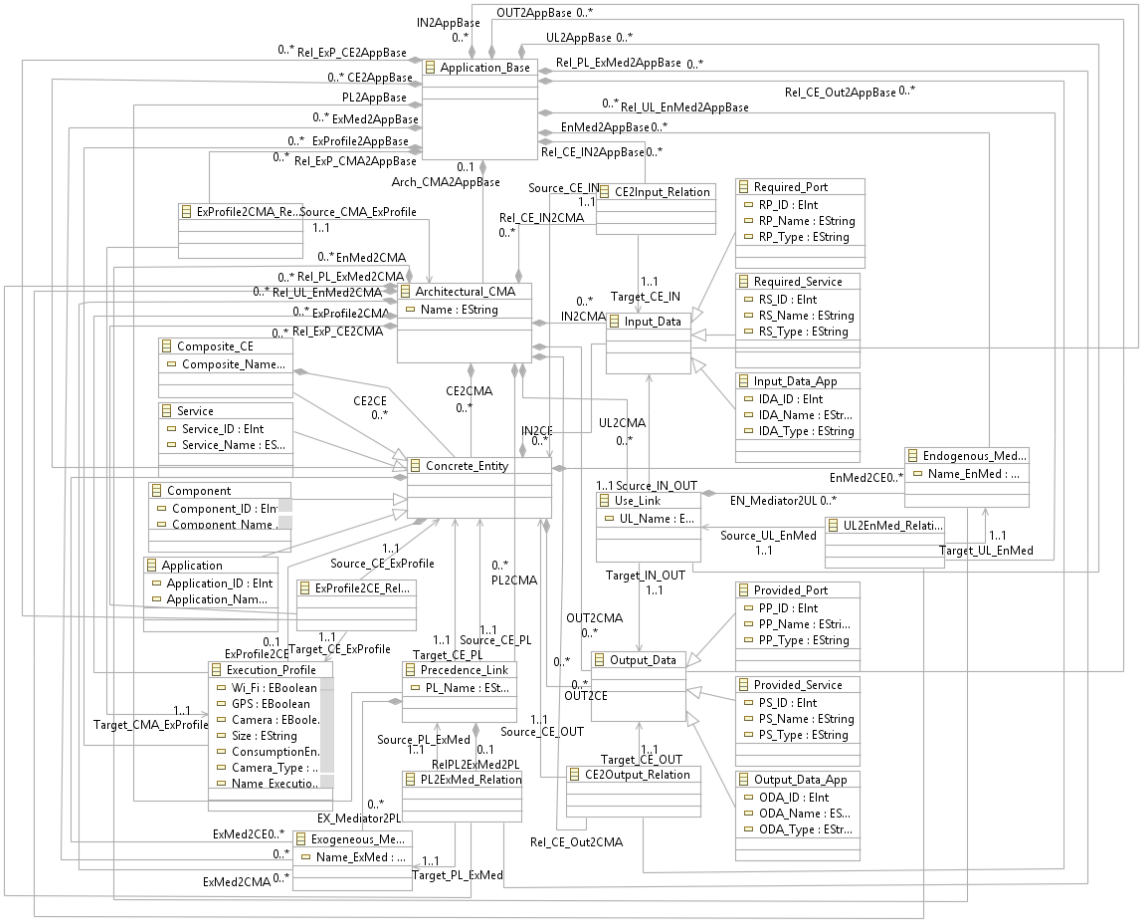

Fig. 9. CMA Architectural Metamodel.ecore

Transformation engine 1 needs, as source information, a set of concrete entities which are compatible with the device mobile features to enable it perform its task. In our approach, we assume that the contextual concrete entities which are required to implement the identified functionalities are already selected and all necessary information about them is expressed in an XML file. To deal with this issue and due to the fact that ATL does not support the parameterization of passage rules (i.e. the lack of the genericity notion) [17]; we have defined a *parameter metamodel* as illustrated in the Fig 10. This latter includes all the necessary values needed to execute the defined passage rules.

However, the different passage rules are written once for all where in each of them we defined a set of ATL *helpers* that serve to extract the needed information from the attributes presented in the parameter model (i.e. instance of parameter metamodel in an XMI format) or to perform various tests. The result of the *transformation engine* 1 is a functional model designed for a specific mobile device context (i.e. refined functional model).

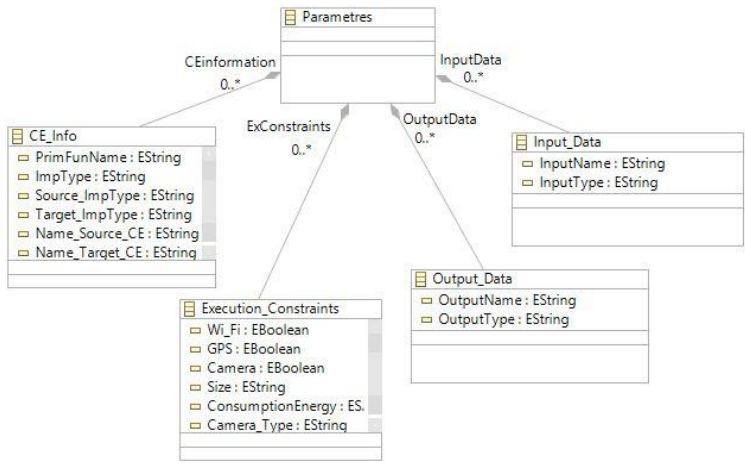

Fig. 10. Parameter Metamodel.ecore

In contrary of *transformation engine* 2, we use CMA functional metamodel as source metamodel and the CMA metamodel as target metamodel. It corresponds to exogenous passage as illustrated in the Listing 1.b. *Transformation engine* 2 aims to exploit the obtained CMA refined functional model in order to generate the CMA architectural model.

```
-- @path
MM=/Functional_M2RefinedFunctional_M/MetaMode
ls/ Functional-CMA-Metamodel.ecore
-- @path
CEinformation=/Functional_M2RefinedFunctional
_M/MetaModels/ParametreMetaModel.ecore
module AddExConstraints;
create RefinedFunctionalModel: MM refining
FunctionalModel : MM,
Parameter : CEinformation;
```

Listing 1.a - Endogenous tranformation (Transformation engine 1)

```
-- @path
ArchCMAMetamodel=/Refined_Fun_Model2Architect
ural_CMA/Metamodels/Architectural-CMA-
Metamodel.ecore
-- @path
RefinedFunMetamodel=/Refined_Fun_Model2Archit
ectural_CMA/Metamodels/Refined-Functional-
CMA-Metamodel.ecore
module RefinedFun2ArchCMA;
create ArchCMAModel : ArchCMAMetamodel from
RefinedFunModel : RefinedFunMetamodel;
```

Listing 1.b - Exogenous tranformation (Transformation engine 2)

Listing 1. Passage rules in ATL language

Our composition mechanism needs implicit triggers of these passage rules in a specific order. The needed passage rules and their invocation order are identified according to the different composition constraints. Therefore, *transformation engine* 2 must respect a set of composition constraints to can achieve its goal. An example of composition constraint is: "composing the Fun 1 that is implemented with a component with the Fun 2 that is implemented with a service needs to associate the precedence link with *exogenous mediators* in order to eliminate the communication heterogeneity problem between these composed concrete entities". For this reason, once the passage rules are written, we convert them to a java code in order to exploit them in the release the composition task (i.e. in order to can trigger them implicitly). From the generated java code of each passage rule, we need only to manipulate the code part shown in the Listing 2 to invoke the corresponding passage rule.

```
PassageRuleName Runner = new PassageRuleName ();
      Runner.loadModels(Refined-Functional-Model, Parameter-Model);
      Runner.doAddImpType(new NullProgressMonitor());
      Runner.saveModels(Architectural-Model);
```

Listing 2. Java code of a *Passage rule*

In the end, the proposed composition mechanism designed to generate for each CMA its own execution profile while triggering *Add-Execution-Profile* rule. It has the potential to fulfill the CMA execution profile according to the different execution constraints of their constituents (i.e. based-on the different execution profiles of the composed entities).

5. Results and discussion

First, the former of the app must focus on the definition of the functional model of the ShopReview App. This functional model is instantiated from the CMA functional metamodel defined previously. The collaboration schema of the desired ShopReview App will be as illustrated in the Fig. 11.

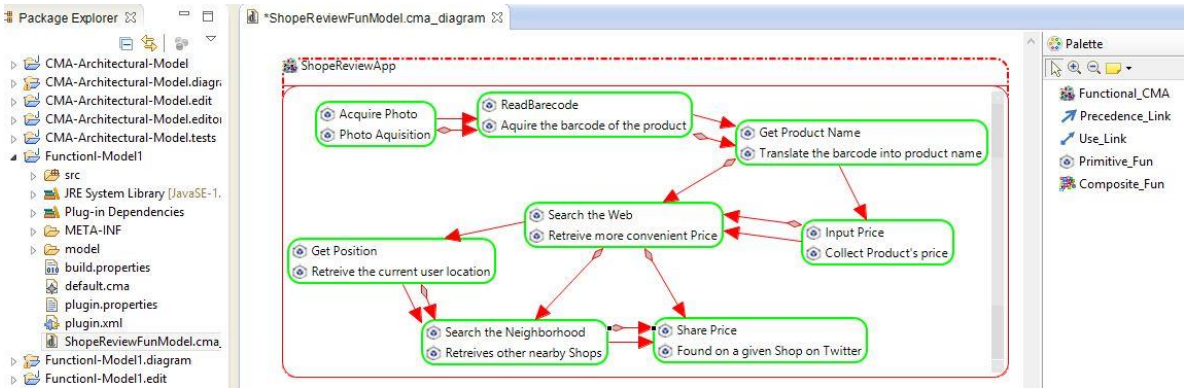

Fig. 11. ShopReview App Functional Model

Now, in order to have the ShopReview App toward a specific mobile environment we must provide the different contextual concrete entities (i.e. that are conform to the contextual information of the mobile device) which will be used to implement the required functionalities. We assign the different contextual information attributes in the parameter model, for any identified functionality, with the different following information: Implementation type, execution constraints, input data, and output data. Afterward, *transformation engine* 1 triggers implicitly the execution of the different necessary passage rules (e.g. AddImpType, AddExConstraints…etc.). The Listing 3 represents a passage rule that aims to attach the functionality *GetPosition* with the implementation type of the contextual concrete entity that will be used to implement it. This passage rule contains two helpers, the first one aims to extract the name of the functionality, for which we need to add the implementation type, from the parameter model in order to select it in the functional model. Whereas the second one aims to extract the implementation type of this functionality from the parameter model and put it in a variable which will be used in this rule to add the implementation type information.

```
-- @path MM=/Functional_M2RefinedFunctional_M/MetaModels/Functional-
RefinedFunctional-CMA-Metamodel.ecore
-- @path CEinformation=/Functional_M2RefinedFunctional_M/MetaModels/
ParametreMetaModel.ecore
module AddImpType;
create OUT : MM refining IN : MM, Parameter : CEinformation;
-- Select the Fun in which we will add the Implementation Type --
helper context MM!Primitive_Fun def : TestPrimFun  : Boolean = let x : String =
CEinformation!CE_Info.allInstances()->collect(p| p.PrimFunName)->first() in
if(self.PrimFunName=x)then true else false endif;
-- Add Implimentation Type for this Functionnality --
 helper  def : ImpType: String =CEinformation!CE_Info.allInstances()->collect(p|
                                  p.ImpType);
```

Listing 3. *Add-Implementation-Type* rule

The result of this task will be a refined functional model of the desired ShopReview App as pictorial in the Fig. 12.

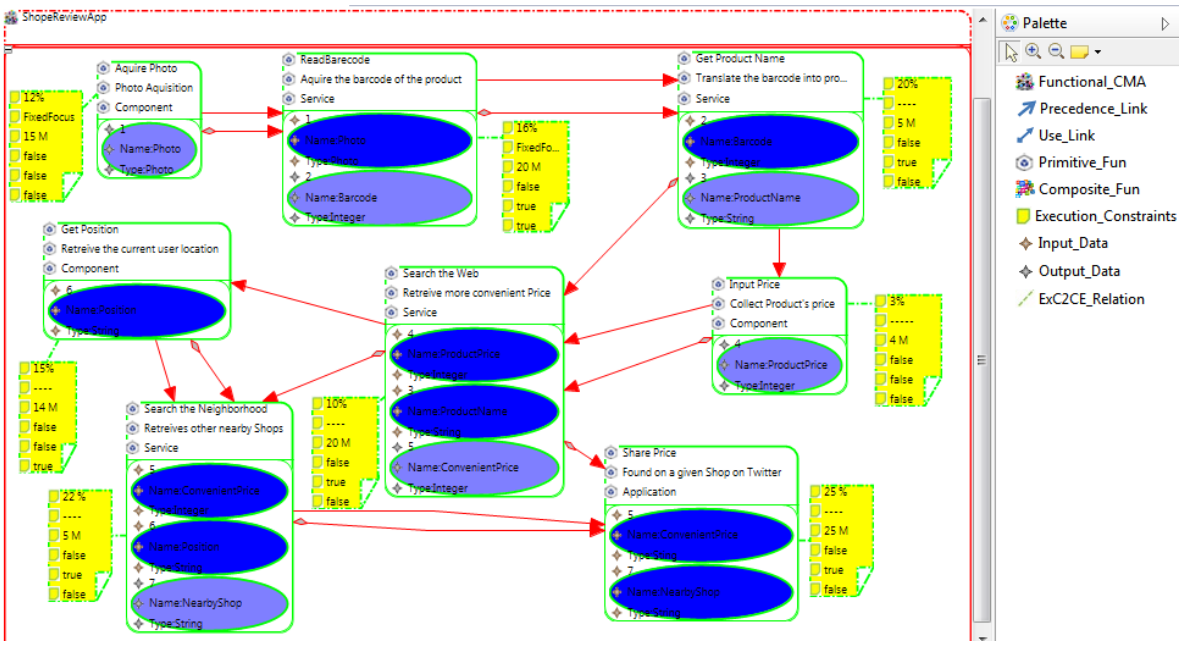

Fig. 12. ShopReview App refined functional model

In this example, we are not illustrating the real values of the different execution constraints of concrete entities but only their relative difference. After generating the ShopReview refined functional model, *transformation engine* 2 triggers implicitly the execution of the necessary passage rules in a specific order to obtain the ShopReview detailed architecture while respecting the different composition constraints presented previously (see Fig. 13). An example of passage rules presented from the *transformation engine 2* is: *RefinedFun2ArchCMA:* which aims to replace each functionality with its corresponding contextual concrete entity; *AddPrecedenceLink_Ex_Med*: which aims to add a precedence link associated with an exogenous mediator between tow concrete entities in order to indicate and manage the heterogeneity point in this composition relation . *AddPrecedenceLink*: aims to add just a precedence link between tow concrete entities which proves that there is no heterogeneity and that these two entities are of the same type. The different necessary passage rules to obtain the architectural model of the ShopReview app and its execution order are defined by the *transformation engine 2* as follow:

- *RefinedFun2ArchCMA // for replacing each functionality with its correspondent contextual CE.*

----- *managing the precedence links*-------

- *AddPrecedenceLink_Ex_Med // between acquire photo and read barcode*
- *AddPrecedenceLink // between read Barcode and GetProductName*
- *AddPrecedenceLink_Ex_Med // between GetProductNameand InputPrice*
- *AddPrecedenceLink_Ex_Med // between InputPrice and Search the web*
- *AddPrecedenceLink_Ex_Med // between Search the web and GetPosition*
- *AddPrecedenceLink_Ex_Med // between GetPosition and Search the neighborhood*
- *AddPrecedenceLink_Ex_Med //between Search the neighborhood and SharePrice*

The passage rules that are dedicated to manage the use links are defined in the same manner of the precedence link.

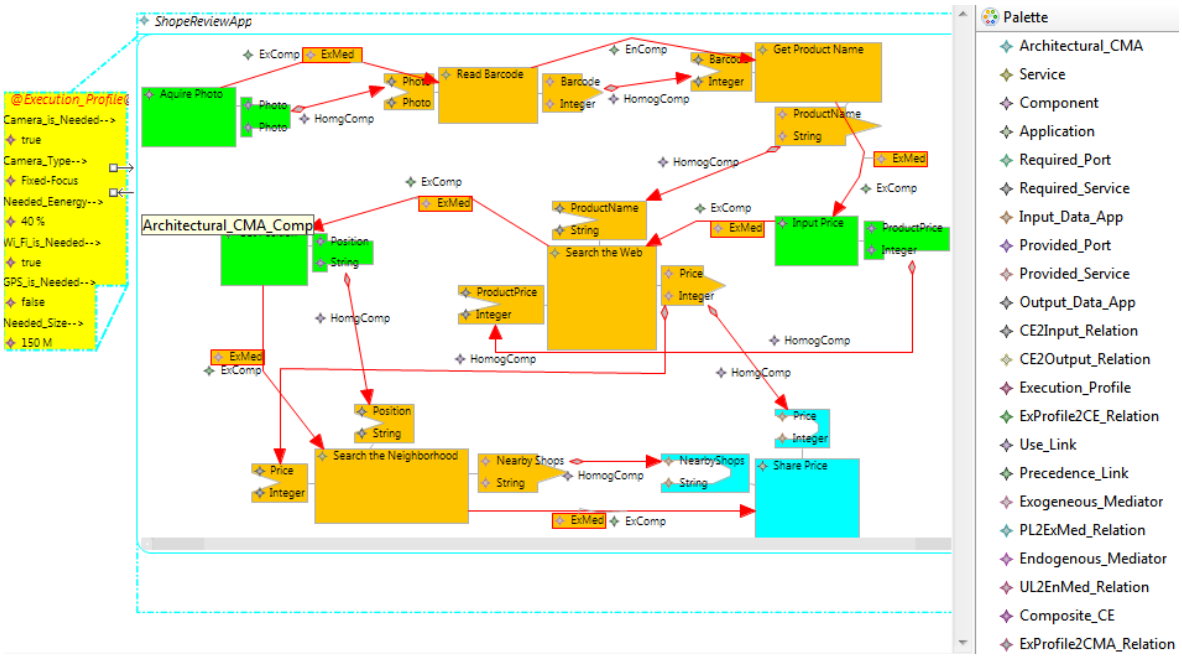

Fig. 13. ShopReview App Architectural model

Our proposed approach allows the composition of adaptive mobile apps starting from the definition of the different needed functionalities until having an adaptive architectural model of the desired CMA including all necessary adaptors. Consequently, it gives us the possibility to obtain several versions of the same desired CMA according to the different contextual information of the mobile devices in which we need deploying our desired app. Differently from traditional composition mechanism that limits the reusability of the obtained desired CMA for a specific context which also cannot be the needed one.

6. Related works

In this section, we describe several approaches that focus on composition mechanism. *Chakraborty* et *al.* [18] describe some issues related to services composition in mobile environment. They presented a distributed Service Composition Protocol for mobile environments that take into consideration mobility, dynamic changing service topology and device resources. They mainly concentrated on a distributed architecture to facilitate the task of composition, but they have neither focused on the application layer or on its adaptation capabilities. In [19], authors presented an automatic approach for web service composition, while addressing the problem of process heterogeneities and data heterogeneities by using a planner and a data mediator. On the contrary, we were meant in [20] to reify the relevant notions of existing mechanisms of composition in a composite service metamodel. This metamodel defines all interlaced features and provides a global and explicit vision of the service composition. Moreover, this approach allows the specification of the auto-composition process: the composite's ability to dynamically modify its architecture and its composition logics according to the environmental context. In [21], authors created a software infrastructure called App-Spotter that makes possible the dynamic and automated selection and composition of software components for building mobile apps. They proposed a mechanism for selection of software components is based-on mobile device features. In [6], authors presented a model to design context-aware services that can be exploited as a flexible domain to automatically generate context-aware compositions by means of a specific tool. In addition, the tool exploits an extension of the problem definition that includes also the context representation for extending the services.

Each of these approaches tries to solve some particular problems from different points of view: composition in terms of components, services, apps or heterogeneous software entities, environment of execution: mobile devices or other (e.g. laptops), adaptation capabilities in terms of composing context-aware or not adaptive apps, composition approach definition at a high level and/or run-time level as illustrated in Table 2.

This multitude of approaches with their features, often specialized, does not have a global vision of mobile apps composition. Our research work intended to clearly express the relevant notions of these existing mechanisms in a composition mechanism for mobile apps using metamodeling approach. Therefore, this humble work represents the first attempt to support the development of context-aware mobile apps by composing heterogeneous software entities at architectural level. So, this composition obeyed to the following specific constraints at architectural level:

- Consider the limited resources offered by mobile devices in order to build adaptive mobile apps.
- Reuse and compose any kind of software entities in order to take profit of existing entities regardless their implementation platform.

	Composition Object			Composition type		Adaption Capabilities		Approach definition at:	
	Components	Services	Heterogeneous entities	Micro (Mobile environment)	Macro (classical environment)	Context-aware	Not-adaptive	High level	Run-time level
chakraborty et al. (2005) [19]		×		×		×		×	
Zixin et al. (2007) [20]		×			×		×	×	×
Anthony and Oussalah (2010) [21]		×			×	×		×	
Ricardo and Vicente (2011) [22]	×			×		×			×
Furno et al. (2014) [7]		×			×	×		×	×
Our Composition Process	×	×	×	×		×		×	

Table 2. Comparison of some existing composition approaches

7. Conclusion

In this paper we have proposed a driven-context composition mechanism of mobile apps by following a metamodeling approach. In this research work, we have dealt and treated the heterogeneity challenges during the composition task (i.e. software entities heterogeneity and mobile devices heterogeneity). We are demonstrated our proposed approach through an example inspired by an existing worldwide distributed mobile app.

Our composition mechanism has the potential to deploy and migrate the same CMA between different mobile platforms. Based-on MDE mechanisms and especially using *model-to-code* transformation mechanism we intend, as a future work, to generate the concrete CMA (i.e. application code) towards a specific platform from the obtained architectural model presented in this work.

More importantly, the mobile device context is temporary and some of its dimensions can be changed frequently. To deal with this issue, we intend also to make our composition mechanism *auto-adaptable* which aims to compose mobile apps that can sense and adapt with every change in their contextual information (i.e. recomposing the mobile app after any change in the mobile context).

References

[1] Ickin, Selim, et al. "Factors influencing quality of experience of commonly used mobile applications." *Communications Magazine, IEEE* 50.4 (2012): 48-56.

[2] Jeffrey S. Hammond and Julie A. Ask , The Future Of Mobile Application Developmentn, January 17, 2013.

[3] Zhang, Dongsong, and Boonlit Adipat. "Challenges, methodologies, and issues in the usability testing of mobile applications." *International Journal of Human-Computer Interaction* 18.3 (2005): 293 308.

[4] Cugola, G., et al., Selfmotion: A declarative approach for adaptive service-oriented mobile applications. Journal of Systems and Software, 2013.

[5] Hock-Koon, A. and M. Oussalah. Expliciting a composite service by a metamodeling approach. in Research Challenges in Information Science (RCIS), 2010 Fourth International Conference on. 2010. IEEE.

[6] Furno, A. and E. Zimeo, Context-aware Composition of Semantic Web Services. Mobile Networks and Applications, 2014. 19(2): p. 235-248.

[7] Hock-Koon, A. and M. Oussalah, Composite service metamodel and auto composition. Journal of Computational Methods in Science and Engineering, 2010. 10: p. 215-229.

[8] Abdelkrim Amirat, Hock-Koon Anthony, and Mourad Chabane Oussalah, Chapter 1: Object-Oriented, Component-Based, Agent-Oriented and Service Oriented Paradigms, In M. C. Oussalah, editor, Software Architectures 1, Wiley-ISTE, April 2014.

[9] Barbier, Eric Cariou—Nicolas Belloir—Franck. " Contracts of transformation for model refinement validation. "IDM 2009 Acts of 5th Days on Model Driven Engineering (2009): 1.

[10]cCzarnecki, Krzysztof, and Simon Helsen. "Feature-based survey of model transformation approaches." IBM Systems Journal 45.3 (2006): 621-645.

[11] Zhang, X., et al., Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. Mobile Networks and Applications, 2011. 16(3): p. 270-284.

[12] Djeddar, A., Bendjenna, H., Amirat, A., & Oussalah, M. (2014, December). Composition Process Based on Heterogeneous Software Entities for Mobile Applications. In Advanced Computer Science Applications and Technologies (ACSAT), 2014 3rd International Conference on (pp. 113-118). IEEE.

[13] Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M., EMF: eclipse modeling framework, Pearson Education, 2008.

[14] Biermann, E., Ehrig, K., Köhler, C., Kuhns, G., Taentzer, G., and Weiss, E., "Graphical definition of in-place transformations in the eclipse modeling framework", In *Model Driven Engineering Languages and Systems*, pp. 425-439, Springer Berlin Heidelberg, 2006.

[15] T. Mens, K. Czarnecki, and P. V. Gorp. 04101 discussion a taxonomy of model transformations. In J. Bezivin and R. Heckel, editors, Language Engineering for Model-Driven Software Development, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum f¨ur Informatik (IBFI), Schloss Dagstuhl, Germany.

[16] Eclipse. Atl project. 2009. http://www.eclipse.org/m2m/atl/.

[17] Amirat, A., Djeddar, A., & Oussalah, M. Evolving and Versioning Software Architectures Using ATL Transformations. The International Arab Conference on Information Technology. University of Nizwa, Oman (ACIT'2014).

[18] Chakraborty, D., et al., Service composition for mobile environments. Mobile Networks and Applications, 2005. 10(4): p. 435-451.

[19] Wu, Z., et al., Automatic composition of semantic web services using process and data mediation. 2007.

[20] Hock-Koon, A. and M. Oussalah, Composite service metamodel and auto composition. Journal of Computational Methods in Science and Engineering, 2010. 10: p. 215-229.

[21] Rosa, R.E.V. and V.F. Lucena Jr. Smart composition of reusable software components in mobile application product lines. In Proceedings of the 2nd International Workshop on Product Line Approaches in Software Engineering. 2011. ACM.