A Smart City Application Modeling Framework: A case study on re-engineering a Smart Retail Platform

Paraskevi Smiari
Department of Informatics and Telecommunications Engineering
University of Western Macedonia,

ImpediMed Hellas,
Greece
psmiari@uowm.gr

Stamatia Bibi
Department of Informatics and Telecommunications Engineering
University of Western Macedonia
Greece
sbibi@uowm.gr

Smart City Application Engineering is a challenging task due to the constantly evolving environment in which these applications operate and the variability of the different types of technologies that synthesize them. Therefore, flexibility and extendibility are two important quality attributes that should be taken into consideration when designing Smart City Applications. In this paper, we propose the Smart City Application Modeling Framework (SCAMF) for analyzing and designing Smart City applications that is based on the concept of Clean Architecture and adopts the representation formalism of feature models. SCAMF methodology is evaluated through a case study on a Smart Retail Platform. Quality indices like flexibility, extendibility along with metrics as complexity, cohesion and design size are compared to the initial version of the application that was completely re-engineered due to maintenance problems. The results of the study suggest that the proposed methodology improves quality indices like flexibility and extendibility up to 120%.

Keywords—Smart City; Feature models; class diagrams; requirements modeling;

I. INTRODUCTION

In the context of sustainable urban development, Smart Cities are highly competitive ecosystems that provide advanced services to citizens covering every aspect of public and private sector spectrum. The development of Smart City applications is a challenging task mainly due to the combination of different technologies that synthesize them and the constantly evolving environment in which they operate [8]. During the analysis and design of Smart City applications an engineer has to take into consideration the two following facts:

a) The *context of Smart Environments is rapidly changing causing functional system changes, to the associated applications.* This fact appoints the need of developing applications that are open to changes, and easily adoptable to the external environment. Thus, Smart City application design should be *flexible* in order to respond to requirements changes.

b) *Smart City applications have to be consistent with a plethora of different types of stakeholders needs*, leveraging ubiquitous connectivity of heterogeneous devices, interaction with a variety of external sources and processing and collection of massive amount of data from different sources [14]. Thus, *variability* is an important characteristic that we need to capture and handle when designing *extendible* Smart City applications that will be able to support future growth of the system.

Among the methods that have been explored in the context of Smart City application engineering is Feature Models (FM) [19] that proved to be a very efficient method for handling the dimension of variability, incorporating usually two phases the domain engineering and the application engineering phase [15]. This two-phase model requires a "big upfront design" of the whole system before individual functionalities can be engineered [7] a fact that undermines the flexibility aspect. In order to deal with this limitation we suggest SCAMF (Smart City Application Modeling Framework) as a methodology for analyzing and designing Smart City applications based on the concepts of Clean Architecture and enabled by the adoption of feature models in order to improve the extendibility and the flexibility aspects that are recognized as important quality drivers[6], [7]. SCAMF is a methodology that consists of four steps a) the definition of the domain rules b) the definition of the application rules c) the interface, controllers and adapters definition, and d) the framework, external sources and data requirements definition. This methodology offers guidance to engineers to design applications ***that are open to change by isolating the business and application logic from the User Interfaces and the external sources, while it offers effective variation mechanisms to handle the heterogeneity of the different technologies.*** The outcome of SCAMF is a feature-based design of the Smart City application. ***As a next step we propose a set of rules for transforming the feature model into a class diagram***.

In order to evaluate the applicability of SCAMF, we present a case study performed on re-engineering a Smart Retail Platform. SCAMF validation was made by comparing the initial version of the Smart Retail Platform to the re-engineered one based on two quality indices a) the level to which the application design can easily support the addition of new functionalities, namely the *extendibility* b) the level to which the application design can confront to changes of the external environment in which it operates, namely the *flexibility*.

The rest of the paper is organized as follows: Section II provides an overview of the related work, Section III presents the SCAMF methodology, Section IV exemplifies SCAMF methodology on the Smart Retail Platform,

Section V presents the case study performed while sections VI and VII discuss the results and conclude the paper.

II. Related Work

In this section we provide an overview of studies that are found in literature so far that focus on requirement analysis and design methods for handling variability and flexibility issues relevant to Smart City applications. Intense research activity has been observed in the field of variability modeling [4], [11] regarding Smart Cities.

Special attention has been paid in dynamic software product lines and their ability to change at runtime with configurations [4] through the usage of feature models. In this study reconfiguration of feature models was done through modeling management operations that used the OSGi framework. Another variability modeling method involving a context-based approach was introduced and applied in a windfarm use case scenario [11]. This approach took advantage of base models and fragments by using an executable processing language such as the Object Management Group standard (OMG). Studies [12] and [13] also chose a fragment-based approach that reuses variants that models variability separately from a base model. Activity Designer was used for base model and fragments specifications and Clafer [12], [13], a class modeling language, was used for modeling variability of things.

Features are explored as a solution to tracing variability in both external and internal architecture and although feature modeling [6], [7], [4], [11], [21], [5] approaches are helpful when it comes to determining the qualitative relationships among the non-functional attributes there have been some observations of a gap that exists between feature modeling and physical modeling [21]. In [5] the authors proposed an extended version of Feature Models, called business process Feature Model (bpFM) so as to confront this gap by using a set of mapping rules from bpFM to business and application processes. A framework that also focuses on flexibility is LateVa [13] that has as a goal to enable process variability at runtime. Other approaches employing feature models is the UMRELA which is an abstract model [16] that uses a Domain Specific Model Language for managing the lifecycle of heterogeneous applications. Other requirement elicitation methods that are found in literature within the context of Smart City applications engineering are the goal-oriented [3] approach, the scenario-based [18], and the artifact-based [9] approach. The goal-oriented [3] approach was adopted for modeling complex systems by eliciting the requirements from textual documentation. The scenario-based approach [18] provides support for systematic analysis of threats, vulnerabilities, and security requirements while the artifact-based approach [9] unveiled a gap between the high-level requirements engineering artifacts and the low-level artifacts.

III. SCAMF: Smart City Application Modeling Framework

In this section we present SCAMF methodology for identifying Smart City application requirements that will serve as an input to the feature model. The application modeling framework for identifying the requirements of Smart City applications is based on the concept of Clean Architecture [10] where software applications are considered to consist of 4 layers a) the Enterprise Business rules b) the Application Business rules c) the Interface Adapters and d) the Frameworks and Drivers. The Dependency Rule is the key idea behind Clean Architecture, where dependencies are encapsulated in each layer of the architecture model, considering that each layer is aware only of the inward one (for example the Interface Adapter layer is only aware of the Application Business Rules Layer). The proposed Smart City application modeling framework is presented in Fig. 1 and consists of the following four main activities:

A. Enterprise business rules definition

The first activity performed is the definition of the enterprise business rules. This activity elicits the requirements relevant to the Enterprise Business Rules layer where our target is to encapsulate the most general and high-level rules [10], that are the least likely to be affected by an operational change, also known as the domain logic. The stakeholder of this activity is the Domain Expert and his responsibility is to produce the Domain model of the initiative. In order to achieve that, the Domain expert has to identify all the objectives and capture the vision of the application by determining general entities referring to a) the Domain rules b) the Policy and the Regulatory restrictions. Entities in that activity encapsulate wide business rules that can be transformed into software artifacts of various levels of granularity from classes to a set of data structures and functions.

B. Application business rules definition

This activity concerns the specification of the application business rules. During this phase the software engineer is expected to conceptualize the target system and define the basic functional and non- functional application requirements. The requirements of Smart City applications are suggested to be analytically recorded into the Use Case diagram and the Use Case Description. These use cases orchestrate the flow of data within the entities defined in this activity and the enterprise entities defined in the previous activity that altogether achieve the goals of the application. Any operational change performed in the application is depicted by changes in the requirements recorded in this step and affect the application entities.

C. Frameworks & Drivers definition

During frameworks and drivers definition we capture the requirements relevant to the frameworks, tools and the persistence layer that remain out of the scope of the application logic and are treated as external resources. During this step we define in parallel the requirements relevant to a) Data management b) Hardware management c) External Systems. The requirements relevant to data management include the definition of data structures and storage types. During hardware definition the Hardware Engineer records all the devices that will be needed for the implementation of the Smart City application such as networking, sensors, actuators and other application-

specific technologies [20]. The final parallel activity involves the definition of the requirements relevant to the External Systems that interact with the application, such systems may provide information to the application or use information from the application.

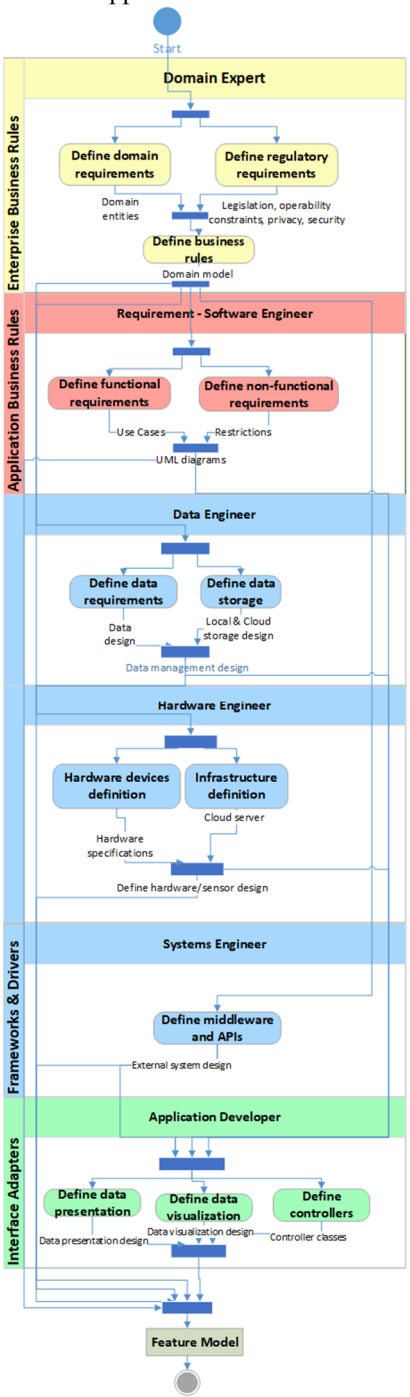

Fig. 1 Smart City application modeling framework

D. *Interface adapters definition*

In this activity we define the requirements of the intermediary layer of the Smart City application that controls the flow of information between the application logic defined in step B and the frameworks, drivers, middleware with which the application interacts defined previously in step C. Responsible for defining the interface adapters is the Application Developer who will define entities relevant a) to the presentation of data (formatting them) , as well as b) to the view of data (routing presentation demands and displaying data to the external sources), and also c) to controlling data, (defining controller entities necessary to convert data from some external form, such as an external service, to the internal form used by the application entities).

IV. The Smart Retail Platform

In this section we exemplify the SCAMF approach defined in the previous section on the Smart Retail Platform. The Smart Retail Platform was developed for a local small-medium Greek supermarket chain consisting of 11 branches all established in one regional unit. The platform was initially designed and implemented two years ago to support the retail business and due to design limitations and the interdependencies between the application logic and the data logic it turned out to be very difficult to apply changes and maintain the application without a major re-engineering. Fig. 2 presents the class diagram of the initial application developed for the Smart Retail Platform. Therefore, the need for re-engineering the application emerged having as a major objective to decouple functionality relevant to the external sources from functionality related to the application logic. In this attempt we adopted the philosophy of Clean Architecture. Another re-engineering objective was to create a main version of the application with all core functionality features where, in the future, optional or alternative features would be easily embedded as variations, without causing structural changes to the main version. For this reason, we selected the feature model approach for recording the requirements. In this section we initially exemplify the SCAMF methodology on the Smart Retail Platform in order to record the requirements of the application in the form of a feature model and then we provide a set of rules for transforming the feature model into a class diagram. In Fig. 3 you can see the use case diagram that describes the re-engineered versions of the platform.

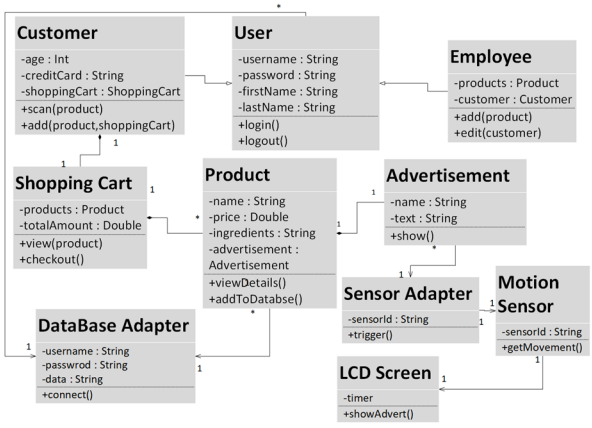

Fig. 2 Initial class diagram

A. *Analyzing the requirements of the Smart Retail Platform*

The SCAMF methodology was applied to record the requirements of the Smart Retail Platform. Starting with the specification of the ***Enterprise business rules***, the Domain Experts that participated in the requirements definition identified as core business entities of the app

four entities: the product, the customer, the advertisement and the retail. As regulatory entities the team recorded taxation withholding regulations and laws relevant to data protection and information privacy. The outcome of this phase was the Domain model along with a set of business rules related to the Smart Retail app.

The next phase incorporates the definition of the ***Application Business Rules***. In the case of the Smart Retail App the software engineers leveraged the power of IoT to gain insights on customer behavior, preferences, determine the efficacy of store displays, optimize floor navigation paths, and perform targeted advertisements zones with personalized coupons and support the 'instant' shopping functionality where the customer can purchase immediately products by scanning them with his mobile phone. The outcome of this phase is the Use Case Diagrams and a set of non- functional requirements.

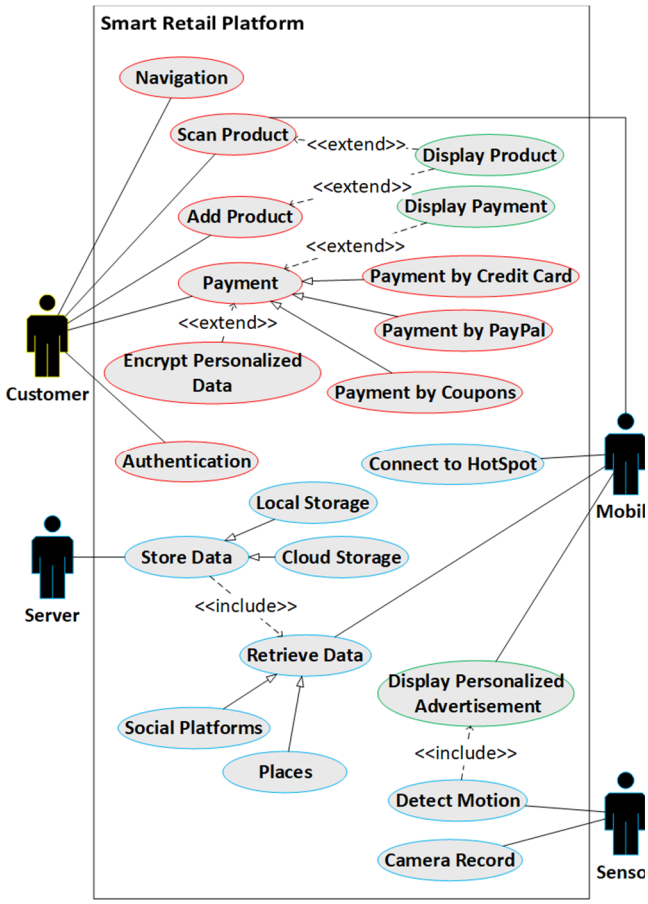

Fig. 3 Use case diagram

The next phase involves the ***Specification of frameworks***, **drivers** and external sources that the app interacts with. Regarding the Data Management requirements, the data engineer along with the owners ended up to an optimized solution of balancing data storage and processing between on premises infrastructure, where sensitive client data are stored, and cloud hosting providers. In parallel, the Hardware Engineer records the relevant infrastructure and devices necessary for the operation of the Smart Retail app. Network infrastructure such as routers, hot spots are required for allowing the communication with the smart devices. Devices like LCD screens, company retail devices, temperature and motion sensors and cameras are the smart devices utilized by the app. Additionally, in this phase the Systems engineer identified all the external systems that interconnected with the Smart Retail App like social platforms APIs that can provide enlightening information regarding the consumer's preferences.

The last phase is the ***definition of the Interface Adapters***. During this phase two types of interface related features were considered. The presenter features that focused on the different UIs designed for the different devices where the application was destined to operate. Additionally, the controller features were defined, that involved the features that would act as intermediates between the application layer and the framework layer, achieving the interconnection among the APIs offered from the external sources and the APIs of the Smart Retail Application layer.

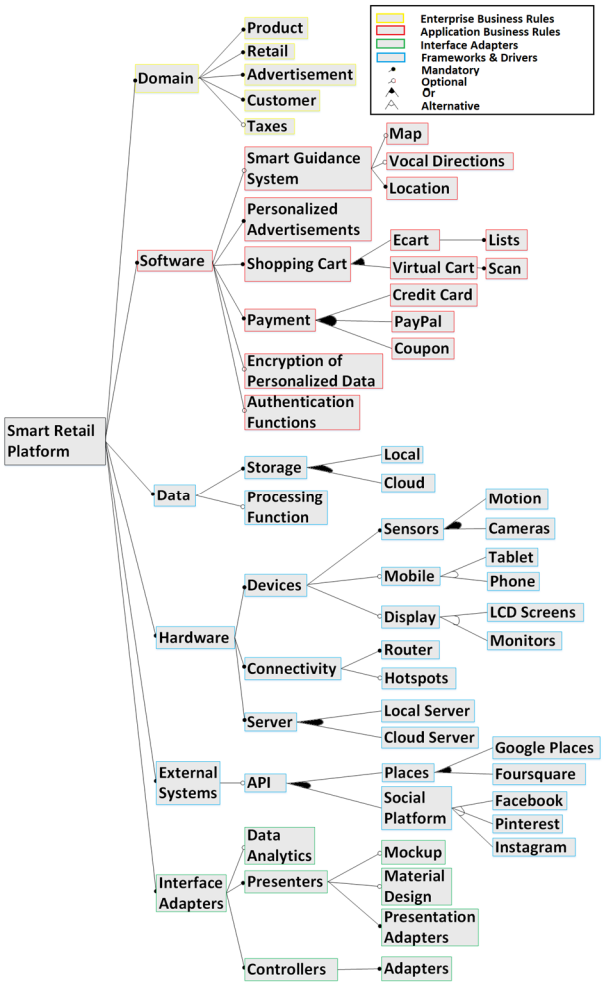

Fig. 4 Feature model

The feature model created for the Smart Retail Platform is presented in Fig. 4.

Features like sensor devices or the retrieval of data are considered mandatory features and must be chosen in all configurations. Data retention or using the API of social platforms are optional features and it's not mandatory to be chosen in all configurations. The children of the Social platforms feature are bind with the OR connection and this

means that at least one child must be chosen. The next step is to define the class diagram. Fig. 6 presents the class diagram related to the "Personalized Advertisement" feature. Each class originates from the four different layers of the application: Enterprise Business Rules, the Application Business Rules, the Interface Adapters or Frameworks& Drivers.

B. Converting the Feature model to Class Diagram

In order to model the relationships of the feature model into a class diagram we applied the following [1] rules:

1) The root and the first level of the diagram are ignored when mapping features to classes as they represent the classification schema of features and not core features
2) The remaining features should correspond to a class with a name exactly like the feature's. For the features that concern hardware or anything other than software when transforming them into classes we consider their relevant API
3) Feature dependencies are transformed into class association with the following stereotypes:
    a) <<mandatory>>
    b) <<optional>>
    c) <<more-of>> (OR)
    d) <<one-of>> (ALTERNATIVE)
4) <<Mandatory>> dependencies are mapped as aggregation (if there are no children with OR, ALTERNATIVE association)
5) <<Optional>> dependencies correspond to an association with a cardinality of 0 or 1
6) The <<one-of>> and <<more-of >> stereotypes for child features result in abstract classes for the parent features, with specific subclasses for each of the alternatives
    a) The one-of dependency for parent features results in a one-to-one association with the root/parent
    b) The more-of dependency between the parent feature and the root/parent result in a one-to-many association with multiplicity equal to the cardinality of the number of or-features
7) Child features (that have OR, ALTERNATIVE dependency) have an inheritance relationship with the parent features
8) Every feature that belongs to Frameworks & Drivers when mapped to a class should always have an adapter class leading it. This adapter will act as a bridge between the Application Business Rules layer and Frameworks & Drivers layer. The adapter classes belong to the Frameworks & Drivers layer and are connected with the corresponding class with direct association with a cardinality of 1. The same connection applies between the specific Frameworks & Drivers adapter and the general Adapter class that belongs to Interface Adapters
9) The highest parent classes of Interface Adapters have a direct association with the highest parent classes of Software with a cardinality of 1

For demonstration purposes we have isolated some parts of the feature model in order to apply the rules mentioned and convert it into a class diagram.

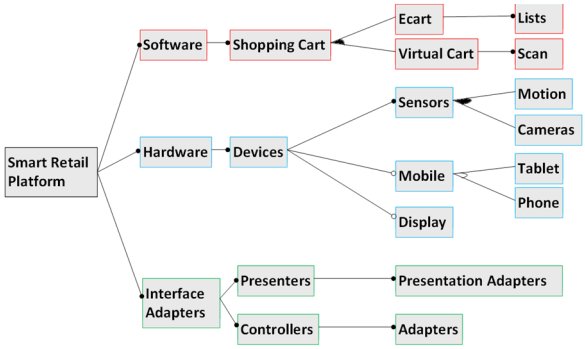

Fig. 5 Feature model shopping cart

According to **rule number 1** features Smart Retail Platform, Software, Hardware, and Interface Adapters should not correspond to a class. By applying **rule number 2** Shopping Cart, eCart, Virtual Cart, Lists, Scan, DeviceAPI, SensorAPI, MotionAPI, CameraAPI, MobileAPI, PhoneAPI, TabletAPI, DisplayAPI, Presenters, Presentation Adapters, Controllers, and Adapters make up the classes for the diagram. Following, features Motion and Cameras have an OR connection with the feature Sensors and according to **rule number 7** Motion and Cameras classes have an inherent relationship with the class Sensors. This also applies to the features Phone and Tablet which have an ALTERNATIVE connection with the feature Mobile, resulting in an inherent relationship with that class. Features Ecart and Virtual Cart have an inherent relationship with Shopping Cart. To determine the dependencies and cardinalities between the classes we apply **rule number 6b** for the feature Sensors. The feature Sensors has a more-of dependency with the root feature Devices so this results in a one-to-many association with multiplicity of 2. As for the feature Mobile we apply **rule number 6a** due to their one-of dependency with the root feature Devices so this results in a one-to-one association. The feature Display corresponds to **rule number 5** so this results to an association with the class Devices with cardinality 0 or 1. Feature Lists has a mandatory dependency with Ecart so by applying **rule number 4** is connected with aggregation dependency with the class Ecart. The same applies between the feature Scan and Virtual Cart as well as Presentation Adapter and Presenter, and Controller and Adapters. According to **rule number 8** we present the feature Adapters in more detail when it comes to mapping it to a class, so we end up with the classes Sensor Adapter, Mobile Adapter, and Display Adapter which have a direct association with the corresponding classes and a cardinality of 1. This also applies to classes ControllerAdapter and Presentation Adapter that have a direct association with the corresponding classes and a cardinality of 1 and an aggregation relationship the Controller and Presenter correspondingly. Finally, by applying **rule number 9** the classes Controller, and Presenter are directly associated with the class Shopping cart with a cardinality of 1.

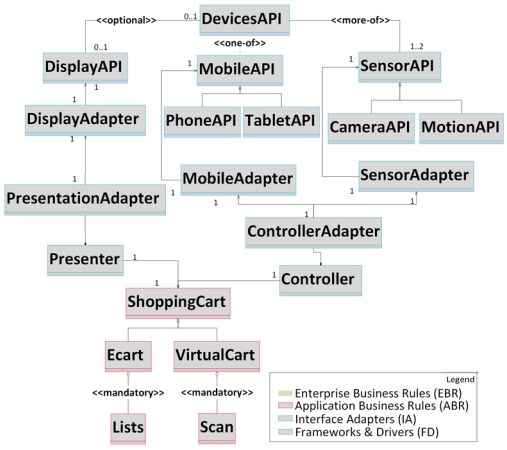

Fig. 6 Re-engineered class diagram

V. Case Study Design and Evaluation

In this section we present the design and the results of the case study performed on re-engineering the Smart Retail Platform based on SCAMF approach, following the guidelines of Runeson et al.[17].

A. Research Objectives and Questions

Our re-engineering target in the Smart Retail Platform was to enable the isolation of business and application logic from frameworks and drivers interacting with the application. The objectives of the re-engineering actions were to: (a) increase the *extendibility* of the application, referring to its ability to easily implement new functionalities by allowing future growth [2] (b) increase the *flexibility* of the application, referring its ability to adapt when external changes occur [1]. Based on the aforementioned objectives, the research questions of this case study are formulated as follows:

RQ1: ***Can SCAMF approach, improve the extendibility of Smart City applications?***

At this research question we aim to validate the proposed design and analysis approach for Smart City Applications with respect to the quality attribute of extendibility. We employed the extendibility quality index suggested by the QMOOD model [2]. We calculated the relevant object-oriented design metrics, presented in TABLE I, and synthesized them to calculate the value of extendibility. This procedure was performed for the two versions of the Smart Retail application, the initial one that was designed adopting ad-hoc procedures and the re-engineered one that employed the suggested methodology.

RQ2: Can SCAMF approach, improve the flexibility of Smart City applications?

In correspondence to the previous research question, here we validate the proposed methodology for analyzing and designing Smart City Applications with respect to the quality attribute of flexibility as defined in the QMOOD model (see TABLE I). The initial and the re-engineered version of the Smart Retail app are compared.

B. Data Collection and Analysis

The data collection process included the calculation of the design metrics presented in TABLE I. These metrics were calculated for the two versions of the Smart Retail application, the initial and the re-engineered one. The design metrics were calculated using the metrics plugin tool for Eclipse (https://github.com/qxo/eclipse-metrics-plugin), for each class of the examined applications. At the end to be able to compare holistically the two applications the mean values of each metric for all the classes participating in the application are taken into account and used to for comparison purposes. TABLE I present the metrics considered within the scope of this study and their description.

TABLE I. Design Metrics

Design Property	Description
Encapsulation: Data Access Metric (DAM)	This metric ranges from 0 to 1 and is calculated as the ratio of the number of private or protected attributes divided by the total declaration of attributes in a class.
Coupling: Direct Class Coupling (DCC)	This metric is calculated as the count of the distinct number of classes that a class is directly related to. Directly means associated by attribute declarations and parameters passing in methods.
Composition: Measure of Aggregation (MOA)	It is calculated as the total number of attribute declarations whose types are user defined classes.
Polymorphism: No. of Polymorphic Methods (NOP)	This metric is calculated as the total count of the methods that can express polymorphic behavior.
Abstraction: Average Number of Ancestors (ANA)	This metric is calculated as the average number of classes from which a class inherits information, it is represented as the number of classes across all paths from the "root" class(es) to all classes in an inheritance structure.
Inheritance: Measure of Functional Abstraction (MFA)	This metric ranges from 0 to 1 and is calculated as the scale of the total methods inherited by a class to the total number of methods accessible by member methods of the class.

The next step is to aggregate the values of the metrics of TABLE I in order to calculate the extendibility index and the flexibility index according to the formula proposed in [2]:

$$\boldsymbol{Flexibility} = \mathbf{0.25} * \boldsymbol{Encapsulation} - \mathbf{0.25} * \boldsymbol{Coupling} + \mathbf{0.5} * \boldsymbol{Composition} + \mathbf{0.5} * \boldsymbol{Polymorphism} \quad (1)$$

$$\boldsymbol{Extendibility} = \mathbf{0.5} * \boldsymbol{Abstraction} - \mathbf{0.5} * \boldsymbol{Coupling} + \mathbf{0.5} * \boldsymbol{Inheritance} + \mathbf{0.5} * \boldsymbol{Polymorphism} \quad (2)$$

VI. Results

In this section, we present the results of the empirical validation of SCAMF. The section is divided into two parts: In Section V.I A, the results of RQ1 regarding the extendibility quality attribute are presented while in Section V.I B the results of RQ2 regarding the flexibility quality attribute are presented. We note that Section VI only presents the raw results of our analysis and answers the research questions. Any interpretation of results and implications to researchers and practitioners are collectively discussed in Section VII.

A. RQ1 - Can SCAMF approach, improve the extendibility of Smart City applications?

TABLE II presents the values of the design metrics that are used to measure the quality properties of extendibility and flexibility. The "SRAv1" column represents the metrics values of the initial version of the Smart Retail application. The value of each metric (apart from DSC) is

calculated as the average number of the value of the metric for all classes participating in the Smart Retail Platform design. The "SRAv2" column presents the metric values addressing the re-engineered design of the Smart Retail application following the SCAMF approach. The metrics that affect the value of extendibility index are: ANA as a measure of abstraction, DCC as a measure of coupling, MFA as a measure of inheritance and NOP as a measure of polymorphism. The results show that properties like abstraction, inheritance and polymorphism present improvement over 50% with the property of MFA presenting an improvement of 126.5%.

TABLE II. DESIGN METRICS RESULTS

	SRAv1	SRAv2	Improvement
DAM	1,00	1,00	0,0%
DCC	1,67	0,82	50,8%
MOA	0,67	0,79	19,2%
NOP	0,50	0,87	74,4%
ANA	0,11	0,17	57,1%
MFA	0,09	0,19	126,5%
Flexibility	0,42	0,88	110,8%
Extendibility	-0,49	0,21	142,9%

Coupling metric, on the other hand, whose value should be controlled and remain relatively low presents a reduction of 50.8% which is actually a very important finding. It seems that SCAMF approach manages to achieve the decoupling of the classes by decomposing functionality to interface and framework classes. Overall the *extendibility* index presents an improvement of 142,9% a fact that indicates the benefits of adopting the SCAMF methodology in the design quality. In the re-engineered version of the Smart Retail Platform inheritance and polymorphism were used as a vehicle to separate high –level modules (Smart Advertisement) to low-level modules (sensors).

B. RQ2 - Can SCAMF approach, improve the flexibility of Smart City applications?

The value of flexibility index is affected by DAM as a measure of encapsulation, DCC as a measure of coupling, MOA as a measure of composition and NOP as a measure of polymorphism. As DCC and NOP metrics also participated in the extendibility index we will now focus on DAM and MOA metrics. DAM presents actually no improvement since in the designers' philosophy, that remained the same during the initial and the re-engineered version of the SRA, was that all attributes within a class should be private or protected. This is the reason that both the initial version and the re-engineered SRA design present 1 as the value of this metric, which is actually the desired value. Regarding MOA we see that there is a small improvement on the level of composition employed in the design that reaches 19%. In the initial version of the SRA the business and application entities of the system and their relationships were identified approaching very closely the real-world scenario were application classes are built from the composition of business classes. Therefore, this part of the design remained the same in the re-engineered solution. The main problem of the initial SRA version was that the business and application logic was mixed-up with the framework constraints. Therefore, in order to separate the different concerns composition was adapted in few cases where the part-whole relationship could be identified (see the relationship of ControllerAdapter and Controller of Fig. 6), though in most cases inheritance was preferred to model entities in the lower level of frameworks and external sources. Overall the *flexibility* index presents an improvement of 110,8% which is a very promising result.

VII. DISCUSSION

In this section we interpret the results obtained by our case study and provide some interesting implications for researchers and practitioners.

A. Interpretation of results

The validation of the SCAMF methodology for analyzing and designing Smart City applications showed that it has the potential to improve the extendibility and the flexibility of the application, two factors that are recognized as important variability drivers [6], [7]. The main advantage of SCAMF methodology is that it is tailored to the specific requirements and standards that smart city applications have to meet. In particular:

- ***SCAMF is independent of the User Interfaces***. A Smart City Application is addressed to a variety of possible users via different technology platforms. Therefore, since it is not possible to have a single running version in all devices and platforms, SCAMF supports the development of applications based on the fact that the application logic remains the same, while the User interfaces through the Presenter entities are the ones that change to address the different platform requirements.
- ***SCAMF is independent of the frameworks and the external sources that interact with the application.*** Since Smart City applications have to interface with many sources of data (sensors, monitors, RFIDs, social media) and store and process data via several means (DBs, files, cloud servers) it is important to ensure that the application is not bound to limitations relevant to the data flow from and to external sources or frameworks. SCAMF supports data – agnostic design of the application isolating the frameworks and the external sources limitations to the Controller entities.
- ***SCAMF offers effective variation mechanisms to handle the constantly evolving environment in which Smart City applications operate.*** SCAMF supports four levels of variations corresponding to the different layers of the Smart City application (the business layer, the application layer, the interface layer and the frameworks layer). Additionally, in each level the variability depicted as alternative, optional or mandatory functionality is expressed with the notation of feature models that is the major modeling technique for capturing operational variations, forcing the isolation of changes in a small fragment of design and code.

B. Implications to reseachers and practitioners.

We encourage researchers to further work on models for designing and analyzing Smart City applications keeping in mind the separation of concerns that in our case study improved several quality characteristics. In that sense analysis and design models should carefully separate the business and application logic that governs the system behavior from the different technologies and frameworks that the application has to comply with. SCAMF methodology can also be further explored and tailored to the needs of particular application domains within the context of a Smart City. Such a customization would offer a better documented, analytical process model for developing applications of a specific type.

Additionally, we encourage researchers to introduce formal metrics and procedures for quantifying SCAMF methodology outputs. In that direction it would be interesting to derive metrics from the derived feature model that will help in providing size and effort estimations regarding the development costs and the design quality.

On the other hand, SCAMF can be a methodological framework in the hands of practitioners so as to guide the process of analyzing and designing Smart City applications, handling their inherent complexity. SCAMF is a very abstract methodology that can be applied in a variety of domains within the context of a Smart City, guiding the analysis and design process of complex applications that need to be extendible, flexible, modular, easy to test and understand.

VIII. CONCLUSIONS

Current practice has shown that the complexity of Smart City applications is increased due to the combination of different types of technology and the need to adapt to the constantly evolving environment. Therefore, engineers should carefully design such applications keeping in my mind that the application will for sure in the future need to interact with systems and technologies not originally designed for, implementing new functionality.

In this study we proposed SCAMF methodology for analyzing and designing Smart City applications that is based on the main concepts of Clean Architecture and employees feature models for documenting the system requirements. The proposed methodology offers a reliable, abstract and easy to understand process for analyzing and designing Smart City applications, that: a) achieves separation of concerns b) handles variations in terms of features. The validation of SCAMF in the re-engineering of a Smart Retail Application showed that the proposed methodology can significantly improve design quality aspects like extendibility and flexibility. Based on these results, we have been able to provide useful implications for researchers and practitioners. As a future work we intend to further document SCAMF methodology and validate it in a variety of application domains.

REFERENCES

[1] K. Bak, Z. Diskin, M. Antkiewicz, K. Czarnecki, and A. Wasowski, “Clafer: unifying class and feature modeling,” Software & Systems Modeling, vol. 15, no. 3, pp. 811–845, Jul. 2016.

[2] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," in *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 4-17, Jan 2002.

[3] E. Casagrande, S. Woldeamlak, W. L. Woon, H. H. Zeineldin and D. Svetinovic, "NLP-KAOS for Systems Goal Elicitation: Smart Metering System Case Study," in *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 941-956, Oct. 1 2014.

[4] C. Cetina, P. Giner, J. Fons and V. Pelechano, "Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes," in *Computer*, vol. 42, no. 10, pp. 37-43, Oct. 2009.

[5] R. Cognini, F. Corradini, A. Polini, and B. Re, "Modelling process intensive scenarios for the smart city." International Conference on Electronic Government. Springer, Berlin, Heidelberg, pp.147-158, 2014.

[6] J. Díaz, J. Pérez, J. Garbajosa, “A Model for Tracing Variability from Features to Product-Line Architectures: A Case Study in Smart Grids” , Requirements Engineering Journal, Springer, vol. 20, pp. 323-343, Sep. 2015.

[7] J. Díaz, J. Pérez, J. Garbajosa, "Agile product-line architecting in practice: A case study in smart grids." *Information and Software Technology,* vol. 56, no. 7, pp. 727-748, 2014.

[8] I. A. T Hashem, et al., “The role of big data in smart city,” Int J Inf Manage, vol. 36, no. 5, pp. 748–758, 2016.

[9] C. Lampasona, P. Diebold, J. Eckhardt, R. Schneider, "Evaluation in practice: artifact-based requirements engineering and scenarios in smart mobility domains." *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2014.

[10] R. C. Martin, “*Clean architecture: a craftsman's guide to software structure and design*.”, Prentice Hall Press, 2017.

[11] A. Murguzur, R. Capilla, S. Trujillo, Ó. Ortiz, and R. E. LopezHerrejon, “Context Variability Modeling for Runtime Configuration of Service-based Dynamic Software Product Lines,” in Proceedings of the 18th International Software Product Line Conference co-located workshops, vol. 2, pp. 2–9, Sep. 2014.

[12] A. Murguzur, X. de Carlos, S. Trujillo and G. Sagardui, "On the support of multi-perspective process models variability for smart environments," *2014 2nd International Conference on Model-Driven Engineering and Software Development*, Portugal, pp. 549-554, 2014.

[13] A. Murguzur, et al., "Runtime Variability for Context-aware Smart Workflows," in *IEEE Software, 2015*.

[14] A. Ojo, E. Curry and F. A. Zeleti, "A tale of open data innovations in five smart cities." *System Sciences (HICSS), 2015 48th Hawaii International Conference on*. IEEE, pp. 2326-2335, 2015.

[15] K. Pohl, G. Böckle, and F. J. Linden,, “Software Product Line Engineering: Foundations, Principles and Techniques.”, SpringerVerlag New York, Inc, 2005.

[16] S. Pradha, A. Dubey, W. R. Otte, G. Karsai, A. Gokhale, “Towards a Product Line of Heterogeneous Distributed Applications.”, ISIS-15-117, Vanderbilt University, 2015.

[17] P. Runeson and M. Host, “Guidelines for conducting and reporting case study research in software engineering”, Empirical Software Engineering, 2009.

[18] H. Suleiman and D. Svetinovic, “Evaluating the effectiveness of the security quality requirements engineering (square) method: a case study using smart grid advanced metering infrastructure,” Requirements Engineering, vol. 18, no. 3, pp. 251–279, 2013.

[19] A. Tahri, L. Duchien, J. Pulou, "Using Feature Models for Distributed Deployment in Extended Smart Home Architecture." *European Conference on Software Architecture*. Springer, pp. 285-293, 2015.

[20] S. Trilles et al., "Deployment of an open sensorized platform in a smart city context", *Future Generation ComputerSystems*, vol. 76, pp. 221-233 Nov. 2016.

[21] A. Venckauskas, V. Stuikys, R. Damasevicius, and N. Jusas, "Modelling of Internet of Things Units for Estimating Security-Energy-Performance Relationships for Quality of Service and Environment Awareness." in Security and Communication Networks 9.16, pp. 3324-3339, 2016.