Towards Integrating Machine Learning Models into Mobile Apps using AppCraft

Lyan Alwakeel[1], Kevin Lano[1] and Hessa Alfraihi[2]

[1]Department of Informatics, King's College London, London, United Kingdom
[2]Department of Information Systems, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University,Riyadh, Saudi Arabia

Abstract
Mobile apps increasingly incorporate machine learning (ML) to enhance their services. However, integrating ML models locally with mobile apps can be challenging. Each ML model has specific designs that accept certain types of input and produce specific outputs. Model-driven engineering (MDE) and low-code solutions can specify the integration process in a high-level language, alleviating this issue. In this paper, we incorporate our framework, AppCraft, with the ML process to generate code for Android and iOS mobile apps with all the necessary components to load the model, process the input data, and display the output results in a user-friendly way. This enhancement contributes to designing and automating the integration of ML engineering processes with mobile apps.

Keywords
Model-Driven Engineering, Low-code, Mobile App, Machine Learning Engineering, Android, iOS

1. Introduction

Mobile apps are widely used features of mobile devices that often use machine learning (ML) to improve their services. On-device inference is a popular example of ML that involves integrating a pre-trained model into a mobile app and performing all inference computations locally on the device. With on-device inference, user data can be processed without being sent to external servers, which enhances privacy and reduces the need for high bandwidth [1]. Furthermore, the pre-trained models used in on-device inference have already been trained on large datasets, making them efficient and effective for intended tasks.

However, integrating ML models into mobile apps can be a challenging task, especially for those with no experience in mobile app development. Each ML model has a specific design that accepts specific types of input and produces specific output. For instance, when training a model and converting it to TensorFlow Lite format for use in a mobile app, the app must be coded to handle input and output tensors at a low-level and associated metadata is necessary to match output values with their intended results. This involves converting input data into a buffer of the underlying data, copying it to the input tensor, invoking the interpreter, and then converting the output tensor data into a usable datatype [2]. The complexity further increases when dealing with more complex data like images, and all these challenges make it difficult to integrate models in mobile apps.

Model-driven engineering (MDE) and Agile/low-code solutions share core principles of abstraction, automation, agility [3], and rapid development, which can address challenges in integrating ML models into mobile apps. We have introduced AppCraft, an Agile MDE framework that includes a Domain-Specific Language (DSL) for mobile apps and two code generators for Android and iOS, enhancing the agility and efficiency of mobile app development. The DSL allows specification of ML process at a high-level of abstraction, to automatically integrate ML models and generate fully functional mobile apps while maintaining quality and reducing costs. Development of complex apps is therefore accelerated, and changing requirements can be rapidly handled by modifying the specification and automatically regenerating the app code.

The remainder of this paper is structured as follows: In Section 2, we discuss the DSL design, followed by case studies in Section 3. Section 4 discusses related work. Finally, we conclude and discuss future work in Section 5.

2. AppCraft Design

The creation of a mobile app utilizing the suggested framework can be broken down into three primary steps, as displayed in Figure 1. To begin with, a modeller writes a declarative textual specification of the app via three dis-

AMDE 2023: Agile Model-driven Engineering Workshop, Part of the Software Technologies: Applications and Foundations (STAF) federated conferences, Eds. K. Lano, H. Alfraihi, S. Rahimi and J. Troya, 20 July 2023, Leicester, UK.
lyan.alwakeel@kcl.ac.uk (L. Alwakeel); kevin.lano@kcl.ac.uk (K. Lano); haalfraihi@pnu.edu.sa (H. Alfraihi)
0000-0003-3779-9939 (L. Alwakeel); 0000-0002-9706-1410 (K. Lano); 0000-0001-8169-3766 (H. Alfraihi)

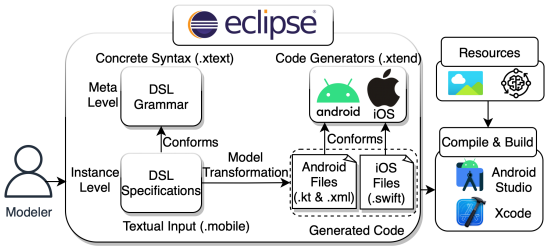

Figure 1: The process of app development with AppCraft framework.

tinct models: data, user interface (UI), and process. The architecture used by these models is the Model-View-Controller (MVC). The data model is defined by a class diagram and adheres to the KM3 [4] meta-model with some modifications. The UI model defines UI components such as buttons and labels, and the process model is described by an activity diagram (expressed as pseudo-code) to detail the actions of the created app. The modeller can specify all three models, as used in [5], or use predefined functions to automatically generate UI, such as CRUD (create, read, update, delete), as well as search. Both specifications can include customized behavior using different activity statements, such as creation, conditional, loop, call, and return. The specification of the three models includes binding statements that link data models with UI models. These bindings link each screen to its corresponding use case, associate each user action with a particular function, and relate UI components to specific attributes.

Secondly, both target platforms' implementation is automatically generated. The specified models are utilized by each code generator to generate platform-specific native code, specifically for Android and iOS. The Android code generator creates Kotlin and XML files, while the iOS generator produces Swift and SwiftUI files. The code generators employed in AppCraft utilize template-based generation, leveraging predefined templates that define the structure and syntax of the generated code. These templates incorporate placeholders that are filled with specific values derived from the DSL models.

Thirdly, the resulting source code of the apps is directly built and compiled on the targeted Integrated Development Environment (IDE) for Android Studio and Xcode for iOS without any modifications, with the ability to add app resources such as images and machine learning models as well as prepare the project with required configuration in case of cloud and ML services. This includes creating cloud database on Firebase, add the configuration file to the project, and add third party components (CocoaPods) to iOS projects. Because the app produced utilizes native platform programming language and UI components, it performs similarly to an app developed manually.

The generated apps follow the generalised MVC/VIPER (Model-View-Controller/View- Interactor-Presenter-Entity-Router) architecture with respect to Clean Architecture principles, which is an architecture that can be applied to both platforms Android and iOS and cover the whole apps.

To specify the ML process, the modeler can specify the name and type of the model, as well as input and expected output of the model in high-level language as listed in Figure 2 and AppCraft will automatically generate fully functionally mobile app.

```
MachineLearning:
 {MachineLearning} 'machineLearning:'
 'modelName:' modelName=ID
 'type:' type = ('audio'|'image'|'numData'|'sensorData')
 ('input:' (input+=Expression (',' input+=Expression)*)?)?
 ('vectorSize:' size1= INT '*' size2= INT)?
 'output:' (output+=ID (','output+=ID)*)  ';';
```

Figure 2: Specification of a machine learning process in AppCraft DSL.

The detailed explanations of the concepts depicted in Figure 2 are as follows:

- **ModelName:** This field represents the name assigned to the model intended for integration.
- **Type:** This is employed to identify the data type to be processed by the integrated model.
- **Input:** When the modeler selects the "numData" type, this field is utilized to specify the input parameter for the model. The modeler can define the input and apply data normalization if necessary, following a similar process employed during training.
- **VectorSize:** This is commonly employed when the modeler selects the "sensorData" type. It allows for the specification of a vector input, which is necessary when dealing with data collected from sensors and requires similar division as applied during the model training.
- **Output:** The field is used to determine the expected output generated by the model.

3. Case Studies

We present two case studies that illustrate the different processing techniques available in AppCraft. The selection of apps and processing techniques is based on our Systematic Literature Review [6] that identifies the characteristics and challenges of mobile health apps.

3.1. Breast Cancer Case Study

The first case study involves a BreastCancer prediction app that employs ML to predict whether the user may

have breast cancer. The ML model was built using deep neural network and trained on the BreastCancer Coimbra dataset [7]. To specify the app, AppCraft's DSL was utilized, which includes a BreastCancer class with the necessary attributes and a persistence stereotype for local saving. The classify function uses the ML process with the "numData" processing type, and input normalization similar to that used during model training, as shown in Figure 3.

```
package breastCancer {
  class BreastCancer {
    stereotype persistent;
    attribute id identity : String; /* principal key */
    attribute age: int;
    attribute bmi: float;
    attribute glucose: float;
    attribute insulin: float;
    attribute homa: float;
    attribute leptin: float;
    attribute adiponectin: float;
    attribute resistin: float;
    attribute mcp: float ;
    attribute outcome derived: String; }

  usecase createBreastCancer (id: String, age: int,
    bmi: float, glucose: float, insulin: float,
    homa: float, leptin: float, adiponectin: float,
    resistin: float, mcp: float, outcome: String
  ) : BreastCancer {
    stereotype create;
    stereotype entity = BreastCancer; }

  usecase listBreastCancer: Sequence(BreastCancer) {
    stereotype list;
    stereotype entity = BreastCancer; }

  usecase classifying(breastCancer: BreastCancer): String {
    stereotype classify;
    stereotype entity = BreastCancer;
    machineLearning:
    modelName: cancer
    type: numData
    input: (ref breastCancer.age - 24) / (89 - 24),
    (ref breastCancer.bmi - 18.37) / (38.578 - 18.37),
    (ref breastCancer.glucose - 60) / (201 - 60),
    (ref breastCancer.insulin - 2.432) / (58.46 - 2.432),
    (ref breastCancer.homa - 0.467) / (25.05 - 0.467),
    (ref breastCancer.leptin - 4.311) / (90.28 - 4.311),
    (ref breastCancer.adiponectin - 1.65) / (38.04 - 1.65),
    (ref breastCancer.resistin- 3.21) / (82.1 - 3.21),
    (ref breastCancer.mcp - 45.84) / (1698.44 - 45.84)
    output: negative, positive;
  }
}
```

Figure 3: Specification of Breast Cancer app.

3.2. Skin Scan Case Study

The second case study involves a SkinScan app that utilizes ML to detect skin conditions. The app was developed using the CRUD functionalities and a cloud stereotype to enable cloud storage, as shown in Figure 4. The "image" processing type was chosen for the ML model, which was trained on the ISIC Archive dataset [8] using the MobileNetV2 architecture.

```
package skincancer {

  class SkinCancer {
    stereotype persistent;

    attribute id identity : String;
    attribute dates : date;
    attribute images : image;
    attribute outcome derived: String;
  }

  usecase crudSkinCancer (id: String, dates: date,
    images: image, outcome: String
  ) : SkinCancer {
    stereotype CRUD;
    stereotype entity = SkinCancer;
  }

  usecase searchSkinCancer (id: String, dates: date,
    images: image, outcome: String
  ) : Sequence(SkinCancer) {
    stereotype searchBy = dates;
    stereotype entity = SkinCancer;
  }

  usecase imgRecognition (skinCancer: SkinCancer): String {
    stereotype classify;
    stereotype entity = SkinCancer;
    machineLearning:
    modelName: SkinCancer
    type: image
    output: benign, malignant;
  }
}
```

Figure 4: Specification of Skin Scan app.

4. Related Work

Various studies have proposed frameworks to generate useful artifacts from models to facilitate mobile app development. These studies differ in scope and modelling language. Some attempts have been made to apply MDE to generate protype that include UI and transitions between pages of mobile apps such as authors in [9] and IFMLEdit [10]. These tools would be more advantageous if they were accompanied by system behavior to generate a comprehensive mobile app. Two closely related frameworks are mD2 [11] and PIMAR [12], which seek to cover both the frontend and backend using DSL. These frameworks and generated apps are based on a MVC architecture and consist of data, UI, and process models and utilized to generate native mobile apps for Android and iOS via Xtext and Xtend. However, they require manual coding for complex functions and are most useful for data-centric apps.

In comparison, our framework is more comprehensive and flexible, allowing for the specification of CRUD functionalities and different behaviours. AppCraft is distinguished by offering 100% generation of fully functional apps that can incorporate ML processes. We use a MVC/VIPER architecture generalized architecture that covers the entire app, rather than just the presentation layer as with MVC. Regarding DSL for ML, there exist languages for modeling specific ML activities, such as Arbiter [13] for ethical ML and DeepDSL [14] for creating

deep learning networks. Additionally, in [15], authors introduce a DSL for modeling ML engineering processes. However, none of these DSLs have been specifically tailored for integrating ML models with mobile apps.

5. Conclusion and Future Work

Combining ML with MDE in an Agile/low-code approach enables rapid development, high-level abstraction, and automation, resulting in ML-enhanced apps that can deliver high-quality results in a shorter timeframe. We believe that AppCraft has the potential to greatly simplify the integration of ML models into mobile apps, and we look forward to exploring its full capabilities and potential in future work.

Moving forward, we plan to evaluate the framework's effectiveness in terms of ease of use, accuracy, and performance. Our goals include measuring the time and effort required to generate an app using AppCraft compared to manual app development, assessing the accuracy of the generated app, as well as enhancing AppCraft by incorporating security and privacy aspects and other types of ML processing, such as audio and video.

References

[1] X. Dai, I. Spasić, S. Chapman, B. Meyer, The state of the art in implementing machine learning for mobile apps: A survey, in: 2020 SoutheastCon, 2020, pp. 1–8.

[2] L. Moroney, AI and machine learning for on-device development, O'Reilly Media, 2021.

[3] A. Bucaioni, A. Cicchetti, F. Ciccozzi, Modelling in low-code development: a multi-vocal systematic review, Software and Systems Modeling 21 (2022) 1959–1981.

[4] F. Jouault, J. Bézivin, Km3: A dsl for metamodel specification, in: Formal Methods for Open Object-Based Distributed Systems, Springer, Berlin, Heidelberg, 2006, pp. 171–185.

[5] L. Alwakeel, K. Lano, Model-driven development of mobile applications, in: ECOOP, 2020.

[6] L. Alwakeel, K. Lano, Functional and technical aspects of self-management mhealth apps: Systematic app search and literature review, JMIR Hum Factors 9 (2022) e29767.

[7] M. Patrício, J. Pereira, J. Crisóstomo, P. Matafome, M. Gomes, R. Seiça, F. Caramelo, Using resistin, glucose, age and BMI to predict the presence of breast cancer, BMC Cancer (2018).

[8] ISIC-archive, International skin imaging collaboration dataset, 2023. URL: https://www.isic-archive.com/#!/topWithHeader/wideContentTop/main, accessed on April 19, 2023.

[9] T. Channonthawat, Y. Limpiyakorn, Model driven development of android application prototypes from windows navigation diagrams, in: ICSN, 2016, pp. 1–4.

[10] C. Bernaschina, S. Comai, P. Fraternali, Online model editing, simulation and code generation for web and mobile applications, MISE '17, IEEE Press, 2017, p. 33–39.

[11] H. Heitkötter, T. A. Majchrzak, H. Kuchen, Cross-platform model-driven development of mobile applications with md2, SAC '13, NY, USA, 2013, p. 526–533.

[12] S. Vaupel, G. Taentzer, J. P. Harries, R. Stroh, R. Gerlach, M. Guckert, Model-driven development of mobile applications allowing role-driven variants, Springer, 2014, pp. 1–17.

[13] J. Zucker, M. d'Leeuwen, Arbiter: A domain-specific language for ethical machine learning, in: The AAAI/ACM Conference on AI, Ethics, and Society, NY, USA, 2020, p. 421–425.

[14] T. Zhao, X. Huang, Design and implementation of deepdsl: A dsl for deep learning, Computer Languages, Systems Structures 54 (2018) 39–70.

[15] S. Morales, R. Clarisó, J. Cabot, Towards a dsl for ai engineering process modeling, in: Product-Focused Software Process Improvement, Springer, Cham, 2022, pp. 53–60.