HealMA: A Model-Driven Framework for Automatic Generation of IoT-Based Android Health Monitoring Applications

Maryam Mehrabi[1], Bahman Zamani[2*] and Abdelwahab Hamou-Lhadj[3]

[1]MDSE Research Group, Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran.
[2*]MDSE Research Group, Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran.
[3]Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada.

*Corresponding author(s). E-mail(s): zamani@eng.ui.ac.ir;
Contributing authors: maryam.mrb94@gmail.com;
wahab.hamou-lhadj@concordia.ca;

Abstract

The development of IoT-based Android health monitoring mobile applications (apps) using traditional software development methods is a challenging task. Developers need to be familiar with various programming languages to manage the heterogeneity of hardware and software systems and to support different communication technologies. To address these problems, in this paper, we first analyze the domain of health monitoring mobile applications and then propose a framework based on model-driven engineering that accelerates the development of such systems. The proposed framework, called HealMA, includes a domain-specific modeling language, a graphical modeling editor, several validation rules, and a set of model-to-code transformations, all packed as an Eclipse plugin. We evaluated the framework to assess its applicability in generating various mobile health applications, as well as its impact on software productivity. To this end, four different health monitoring applications have been automatically generated. Then, we evaluated the productivity of software developers by comparing the time and effort it takes to use HealMA compared to a

code-centric process. As part of the evaluation, we also evaluated the usability of HealMA-generated apps by conducting a user study. The results show that HealMA is both applicable and beneficial for automatic generation of usable IoT-based Android health monitoring apps.

Keywords: Health Monitoring, Android, IoT, Model-Driven Engineering

1 Introduction

Remote health monitoring is an essential component of a healthcare system to address the needs of growing populations located in various geographical areas [1]. This is particularly important when dealing with chronic and infectious diseases (e.g., blood pressure, respiratory diseases, HIV), which require long-term monitoring and treatments [2].

A recent trend in remote health monitoring is to make use of the Internet of Things (IoT) [3], which encompasses a set of technologies that connect various devices anywhere and at any time. A typical application of IoT in health monitoring is to monitor a patient's health status using mobile applications (apps) [4]. This is particularly relevant in today's context because of the increasing number of smartphone users. According to Statista, about 3.8 billion people will have access to a smartphone by the end of 2021 [5]. The health monitoring apps market size was valued at USD40 billion in 2020 and is expected to grow at an annual growth rate of 17.7% from 2021 to 2028 [6].

The development of health monitoring apps from scratch or using domain-independent modeling languages, e.g. UML, is a time-consuming and repetitive task. The challenge is that the developers must have knowledge about the symptoms that need to be monitored, manage the heterogeneity of hardware and software systems, and understand the different communication technologies that are supported in IoT [4, 7–9]. To address this challenge, in this paper, we propose a model-driven framework, called HealMA (Health Monitoring App), for rapid development of remote health monitoring apps. HealMA consists of four components: (1) a domain-specific modeling language (DSML) that supports the concepts of remote health monitoring, which we identified based on a thorough analysis of the domain, (2) a graphical editor that enables developers to model a remote health monitoring app, (3) constraints and validation rules, and (4) a model-to-code transformation engine that generates automatically the final app code from the model. These components are all packaged as an Eclipse plugin. Currently, HealMA supports the development of Android apps only, considering the fact that the Android app market accounts for 72.2% of the mobile app market share in 2021 [10]. The generated apps will be posted on app markets to be used by healthcare patients and providers.

The HealMA framework is very intuitive to use. Once the app requirements are defined, a developer uses the modeling editor to select the model elements relevant to the app by a simple drag and drop. After the model is validated

against predefined constraints, the app code (including Java and XML files) is then generated automatically through our embedded model-to-code transformation engine. The final step is to open the generated code in an Android IDE, complete the project according to the comments provided by the HealMA framework, and customize the project as needed. As we will show in the evaluation section, HealMA simplifies greatly the development process of health monitoring apps, which should result in increased productivity.

We evaluated HealMA using two strategies. First, we implemented four case studies in which we generated four different remote health monitoring apps. The objective is to evaluate the applicability of HealMA in generating different monitoring apps, as well as assess the effectiveness of HealMA in increasing the productivity of developing health monitoring apps (i.e., effort and time-saving in the development) compared to a pure code-centric approach. The second strategy consists of conducting a user study in which we evaluate the usability of HealMA from the developers' perspective. The results show that HealMA can help developers generate health monitoring apps with less effort, and the generated apps are usable for the end users.

The remainder of this paper is structured as follows. In Section 2, we address the background, define the scope of this research, and review the related work. Section 3 introduces the proposed framework, including the modeling language, graphical modeling editor, validation rules, and model-to-code engine. In Section 4, we discuss the evaluation of the proposed framework from different viewpoints. We also address the threats to validity in this section. Finally, we conclude the paper and elaborate on the future work in Section 5.

2 Background and Related Work

This section is divided into two parts. In the first part, we provide background information about model-driven engineering and remote health monitoring using IoT. We also show the role of applications in the IoT-based health monitoring process, and define the scope of this research. In the second part, we discuss about the related work.

2.1 Background

2.1.1 Model-Driven Engineering

Models are a simplified representation of reality. They make complex facts more understandable by raising the level of abstraction. In traditional software development approaches, models were used for design and documentation. In recent years, Model-Driven Engineering (MDE) has emerged as a new software engineering approach where models are key software development assets used to build the system. The code is generated from models. The aim is to reduce development complexity by focusing on models instead of code [11].

In MDE, the system is produced based on different abstraction levels of models and transformations. An example of this can be seen in the Model-Driven Architecture (MDA) standard proposed by the Object Management Group (OMG)[1]. MDA divides the development process into four levels. In the first level, the requirements of the system are defined with the highest abstraction level in a Computational Independent Model (CIM), which could be a business or domain model. In the second level, CIM is transformed into a Platform Independent Model (PIM) that represents the structure and behavior of the system without considering the execution platform. On the third level, the PIM is transformed into a Platform Specific Model (PSM). A PSM is a model of the execution platform. Finally, the PSM model is transformed into code [11].

An important aspect of MDE is the development of Domain-Specific Modeling Languages (DSMLs) to capture the domain concepts and their relationship. DSMLs help design the system. Same as other languages, a DSML includes three main parts, namely an abstract syntax, a concrete syntax, and semantics. The abstract syntax shows the grammar of the language and is defined using a metamodel. The concrete syntax is a graphical representation of the language elements that are defined in the abstract syntax to enable developers to use the language. Semantics capture the meaning of the elements of the metamodel, i.e., the language elements and their relationships [11, 12].

Transformations are an important pillar of any MDE approach. The generation of code requires the definition of transformation rules. A transformation rule is simply a program that takes an artifact as input and creates another artifact as output [13]. Considering 'model' and 'code' as the artifacts, there are four types of transformations: model-to-model, model-to-code, code-to-model, and code-to-code. To automatically generate code from the models using MDE, model-to-code transformations are used [11].

2.1.2 Remote Health Monitoring using IoT

IoT is a new trend that makes various domains smart by connecting different objects. Remote health monitoring is an example of IoT applications in the health domain. IoT helps collect data and track the patient health status, efficiently [4].

There have been many IoT-based remote health monitoring architectures that are proposed in the literature [14–16]. These architectures mainly vary in terms of the devices used for sensing health and environmental parameters, the use of different communication technologies for transmitting data, and the types of applications for data analysis and visualization. Most architectures rely on mobile apps for performing various tasks. Mobile apps may are used by patients to visualize their health status and also as a gateway to transmit health information to healthcare providers. Health professionals use mobile apps and web-based systems to visualize and monitor the health conditions of patients remotely.

[1] http://www.omg.org

Figure 1 shows a typical remote health monitoring architecture which consists of three layers: 1) patient layer, 2) communication and storage layer, and 3) supervisor layer. These layers are explained in the following.

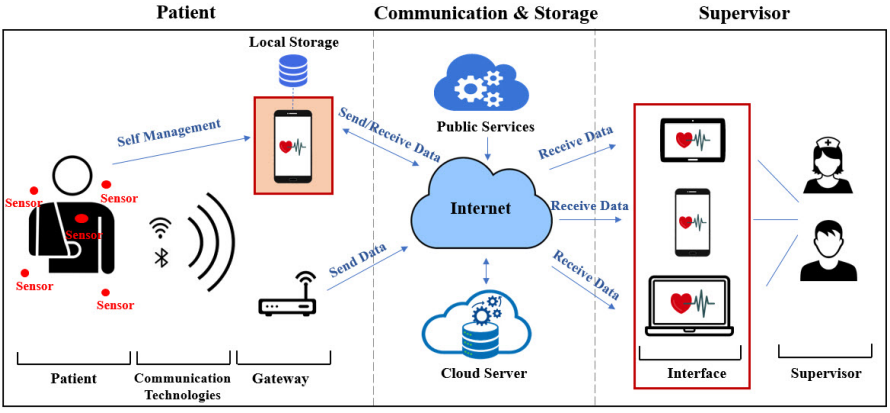

Fig. 1 A typical architecture for a remote health monitoring system (The red boxes show the role of apps in remote monitoring process. The shaded box shows the scope of HealMA)

- The Patient Layer: This layer consists of a set of physical objects (e.g., wearable sensors) that are placed on the patient to collect blood pressure, heartbeat rate, and other health-related measurements of interest. Devices can also be used to record environmental parameters such as humidity and temperature. The collected data is often sent to a gateway device that relays the information to healthcare providers. A typical gateway is the patient's smartphone. The communication technology is selected based on the distance between the objects and the gateway. Bluetooth is used for short-range communications, such as connecting a smartwatch to a smartphone. Wi-Fi is used for longer distances such as collecting the temperature of a room in a building. In addition to being a gateway, the patient's smartphone can do some basic processing, store the data locally, and visualize the results for the patient.
- The Communication and Storage Layer: Gateways send the collected data to the cloud servers through Internet. The data is stored on the servers and ready to be processed. Additionally, in this layer, environmental parameters such as the air quality index (AQI) and the weather status could be accessed by the patients and supervisors through online public services.
- The Supervisor Layer: The results of processing should be reported in an appropriate format for different participants in the health monitoring process, e.g., patients, nurses, doctors, and patient's family members. This is usually done through applications that act as an interface (mobile apps or traditional web applications).

As indicated in the shaded box in Figure 1, the scope of our work (HealMA) is on the automatic generation of the apps that run on the patient's smartphone to act as a gateway and also to provide different functionalities for the patient such as visualizing and managing their health status.

2.2 Related Work

To our knowledge, there is no work that focused primarily on the generation of health monitoring applications using an MDE approach. In this section, we considered a wider range and reviewed studies that focus on conceptual models related to IoT-based health care systems, as well as the studies that use MDE in the domains close to the IoT-based health care domain. In the following, we group these studies into three categories in terms of domain and discuss how they differ from our study.

The first category is concerned with the studies that focused on conceptualization or development of the IoT-based systems and applications. Bauer et al. [17] worked on domain engineering and introduced the IoT-A domain model which is part of a reference architecture, designed as a common ground to show the main components of IoT-based systems. Patel et al. [18] also focused on domain engineering and introduced another domain model for IoT-based applications, which, compared to IoT-A, is expressed in a lower level of abstraction and is able to provide more details about the domain. Ontologies have also been used to model IoT domain concepts, e.g., in the work of Bermudez et. al [19]. Einarsson et al. [20] introduced a DSML called SmartHomeML to automatically generate smart home connectivity services. Hussein et al. [21] used model-driven development to generate automatically Java code for self-adaptive IoT-based systems. The difference between these studies and the work presented in this paper is that these studies cover the broader domain of IoT, whereas our work has a more specific scope of IoT-based health monitoring.

The second category covers studies that focused particularly on modeling IoT-based health monitoring systems. Rhayem et al. [22] extended the ontologies of the IoT-based systems for the health monitoring domain and introduced HealthIoT, an ontology for storing data in a structured way for semantic reasoning on health data. Gomez et al. [23] developed a client/server architecture to monitor the patient health status and to make recommendations to improve the patient's condition. The recommendations are based on an ontology that includes devices, persons, locations, and time parts. Chellouche et al. [24] proposed a framework to address semantic interoperability and automatic decision taking in machine-to-machine (M2M) health care systems. To store health data in a structured way, an ontology-based data model is presented that facilitates reuse and knowledge sharing. Ajami et al. [25] introduced an ontology, named COPDology, for remote monitoring of COPD patients. COPDology represents various aspects of remote health monitoring including patient's clinical information, patient's physical parameters, patient's activities, environmental parameters, disease information, location information, device information, and service information. Considering such detailed information helps do semantic

reasoning more accurately. Although the abstraction level of these studies is the same as our research, the goals are different. The objective of using ontologies is to create a structural model that would allow reasoning tasks whereas our research aims to model the domain of IoT health monitoring apps and provide tools for developers to automatically create these apps using a domain specific language.

The third category of the research studies related to our work is dedicated to the studies that focus on the development of mobile apps using model-driven techniques. Among the works in this category are domain-independent languages such as IFML[2] that have the potential to be used for modeling mobile applications. In some cases, such as the work done by Brambilla et al. [26], the language (IFML) has been extended for the purpose of modeling and automatic generation of mobile apps. Other works in this category introduced a new language to support the automatic generation of mobile apps. Usman et al. [27] proposed a model-driven approach based on feature modeling and UML profiling to generate mobile applications for different platforms such as Android and Windows Phone. Vaupel et al. [28] analyzed the mobile apps domain and presented a feature model, a modeling language, an editor, and a code generation infrastructure for the automatic generation of data-driven applications. Gharaat et al. [29] used a model-driven engineering approach to create geo-location apps. The authors proposed a metamodel that captures the features of geo-location apps from which various apps that support these features can be automatically generated. Nunez et al. [30] proposed a model-driven approach to tackle the complexity of designing the data layer for native mobile applications. This approach supports both online and offline methods for data management. They also provide some transformation rules to generate code for Android and Windows Phone applications. Similar to our research, these works have also focused on mobile app development blue using model-driven techniques. We argue that, generating a health monitoring mobile app using domain-independent languages or languages that are invented for designing all types of mobile apps needs a deep knowledge about health monitoring domain. Therefore, in our work, we focused on a specific domain (i.e., the health monitoring) to reduce the knowledge and expertise which is required for designing a health monitoring app.

To show the main similarities and differences between the current research and related work, we compare them in Table 1. The column "Domain" shows three categories that we used to classify related work. The "Approach" column describes the technique proposed by the studies. The"Purpose" column indicates the objective of the underlying research. The "Achievements" column summarizes the outcome of the research. All the research studies focused on domain analysis by proposing a conceptual model based on the objective of the study. For the studies that used MDE, in addition to the conceptual model, the outcome of the respective approaches are classified as either "Modeling" or "Code Generation tool".

[2] https://www.ifml.org/

To summarize, as indicated in the last row of Table 1, in our work we analyzed the domain of IoT-based health monitoring mobile apps and used MDE for the automatic generation of IoT-based health monitoring mobile apps. To achieve this, we designed a conceptual model (metamodel), developed a modeling editor, and provided transformation rules for automatic code generation.

3 HealMA Framework

HealMA is a model-driven framework for the rapid development of remote health monitoring apps that run on the patient's smartphone. It consists of four main components: 1) a DSML that describes the main concepts of the health monitoring application domain, 2) a graphical modeling editor to make modeling easier, 3) a set of constraint and validation rules to check the validity of the models, and 4) a model-to-code transformation engine to generate the application code (both Java and XML) for the Android platform. In the following Sections (3.1 to 3.4), we discuss these components in more detail. Then in Section 3.6, we present the steps that developers should follow for developing an app using HealMA.

Table 1 Comparison between related work and current research

	Domain			Approach			Purpose			Achievements		
Research	IoT	IoT+Health	Mobile Apps	blueDomain Engineering	MDE	Ontology	Domain Analysis	Development	Reasoning	Conceptual Model	Modeling Tool	Code Generation
Bauer et al. [17]	●	○	○	●	○	○	●	○	○	○	○	○
Bermudez et al. [19]	●	○	○	○	○	●	●	○	●	●	○	○
Patel et al. [18]	●	○	○	●	○	○	●	○	○	●	○	○
Einarsson et al. [20]	●	○	○	○	●	○	●	●	○	●	○	●
Hussein et al. [21]	●	○	○	○	●	○	●	●	○	●	○	●
Rhayem et al. [22]	○	●	○	○	○	●	●	○	●	●	○	○
Gomez et al. [23]	○	●	○	○	○	●	●	○	●	●	○	○
Chellouche et al. [24]	○	●	○	○	○	●	●	○	●	●	○	○
Ajami et al. [25]	○	●	○	○	○	●	●	○	●	●	○	○
Brambilla et al. [26]	○	○	●	○	●	○	○	●	○	●	●	●
Usman et al. [27]	○	○	●	○	●	○	●	●	○	●	○	●
Vaupel et al. [28]	○	○	●	○	●	○	●	●	○	●	●	●
Gharaat et al. [29]	○	○	●	○	●	○	●	●	○	●	●	●
Nunez et al. [30]	○	○	●	○	●	○	●	●	○	●	○	●
Current Research	○	●	●	○	●	○	●	●	○	●	●	●

Supported ● Not Supported ○

3.1 HealMA DSML

In order to define the HealMA DSML, we followed the phases for developing a DSML that were introduced by Mernik et al. [31], namely Decision, Analysis, Design, Implementation, and Deployment. In the first phase, Decision, the necessity of defining the language is discussed. In the second phase, the main concepts of the domain are extracted. The objective of the third phase is to create the metamodel (the abstract syntax) of the DSML by extracting the domain concepts and their relationships. In the fourth phase, the metamodel is implemented in a tool. The last phase consists of defining the concrete syntax of the language and building a graphical editor. In the following, we describe each phase in the context of HealMA.

3.1.1 Decision

The emergence of remote health monitoring has created a demand for developing mobile apps for health monitoring. In general, such apps receive the measured health data and visualize it for the patient. Developing these apps needs expertise in different aspects, including programming skills to communicate with different devices and visualizing the data in an understandable format. Therefore, developing from scratch using the general-purpose languages is a time-consuming and repetitive task, which may delay the time to market. Due to the fact that these apps tend to have similar structure and behavior, defining a new DSML for the health monitoring domain can help capture common concepts. The DSML should be expressive enough to cover a wide range of health measurements such as blood pressure, heart rate, etc. Developers can use this DSML to model concepts specific to their monitoring app.

3.1.2 Domain Analysis

To analyze the domain of health monitoring apps, we followed the feature-oriented domain analysis (FODA) methodology presented by Kang et al. [32], which is a method for classifying features of an application by distinguishing the mandatory and optional features and their relations.

To obtain a feature model for the domain of remote health monitoring apps, we went through the following steps. First, we selected the top 20 highly ranked apps in this domain from the Google Play Store[3] and the Apple Store[4] and examined the concepts they support by installing and running the apps and referring to any available documentation. Table 2 shows the name, number of downloads, and satisfaction rate for the selected apps. Then, we used the research studies discussed in the related work section (see Table 1) as another source of information to uncover important concepts that may have not been implemented in existing apps. Our objective is to cover as many features as

[3] https://play.google.com/store/apps?hl=en
[4] https://www.apple.com/ios/app-store/

possible that would make HealMA a powerful framework for generating different types of health monitoring apps. Based on our analysis of the domain, using the apps and the research papers, we grouped the extracted concepts into 7 classes as follows.

Table 2 Selected apps for feature extraction

No.	App name	Downloads	Satisfaction rate
1	Huawei Health	+100 M	3.6
2	Mi Fit	+50 M	4.6
3	Heart Rate Monitor	+1 M	4.5
4	Heart Rate Plus: Pulse Monitor	+1 M	4.0
5	Diabetes: M-Management & Blood Sugar Track App	+500 K	4.5
6	Blood Pressure Diary	+500 K	4.2
7	Blood Pressure Analyze	+100 K	4.2
8	BP Journal – Blood pressure Diary	+100 K	4.8
9	Blood pressure recorder & bp diary	+100 K	3.9
10	Body Temperature Diary	+100 K	3.7
11	Apple Health	N.A.*	-
12	Instant Heart Rate: HR Monitor	N.A.*	4.9
13	HeartRateLite	N.A.*	4.6
14	BPMonitor	N.A.*	4.6
15	Qardio	N.A.*	4.7
16	Glucose-Blood Sugar Tracker	N.A.*	4.7
17	Fever Tracker	N.A.*	2.9
18	Glucose Buddy Diabetes Tracker	N.A.*	4.8
19	VitalDetect App	N.A.*	5
20	Welltory	N.A.*	4.5

* No. of downloads is "Not Available" in App Store

1. **Data Source:** Health tracking apps receive health data manually or automatically. In the manual way (e.g., in "Blood pressure Diary"), patients themselves should measure the health parameters and enter the data into the app manually. However, some apps (e.g., "Heart Rate Puls: Pulse Monitor") receive data from smartphone internal resources (sensors), automatically. Other apps (e.g., "Huawei Health" and "Mi Fit") receive data from wearable devices such as smartwatches. In addition to the apps, the data could be received from an external device such as a Raspberry Pi 3 board by wireless communication mechanisms (e.g., in [33]). Also, the data could be received from storage and computational services to be visualized for the users (e.g., in [18]).
2. **Platform:** The 20 investigated health tracking apps were developed for different platforms, e.g., Android (apps 1 to 10) and iOS (apps 11 to 20).
3. **Storage:** The received health data could be stored locally on the smartphone or be stored on cloud servers. There are apps (e.g., "Qardio") that just store the data locally, while many other apps (e.g., "HeartRateLite" and "Diabetes: M-Management & Blood Sugar Track App") provide support for both local and cloud storage.

4. **Functionality:** Applications could support different functionalities to improve self-management and supervising the patient.
   - **Authentication (sign up and login):** Most of the apps (e.g., "Glucose Buddy Diabetes Tracker"), provide authentication features to support privacy and security. Some other apps (e.g., "Heart Rate Pulse-Pulse & Heart Rate Monitor") do not support authentication.
   - **Profiling:** Getting personal information (e.g., in "Apple Health"), helps customize the app for the users by giving them better ways to manage the app features.
   - **Analysis:** Simple data processing for analysis (e.g., in "Instant Heart Rate: HR Monitor"), helps to know if the health status is normal.
   - **Alert:** Alerts are used in apps (e.g., "Qardio") for reminding important tasks such as measuring health parameters and showing the measurement results. Alerts could also be used for notifying the supervisor (e.g., in [33]) of potential abnormalities and emergencies.
   - **Advice:** Providing guidelines to control abnormalities (e.g., in "Diabetes: M-Management & Blood Sugar Track App") helps manage emergencies.
   - **Report:** Data visualization and report creation, in different time periods and in a user-friendly format for patients and supervisors, help track the patient health status. Some apps (e.g., "Heart Rate Monitor") provide different types of reports for the patient, while other apps (e.g., "BP Journal – Blood pressure Diary") create reports in different formats to be shared with the supervisor.
   - **Learning:** Some apps (e.g., "Welltory") provide information sources to inform the patient about the disease and appropriate lifestyle during the illness to increase the quality of life.
   - **Treatment:** Some apps (e.g., "Diabetes: M- Management & Blood Sugar Track App") show the process of treatment including medication, activity, and diet to be followed by the patients.
   - **Clinical Info:** The patient's medical info (e.g., in [25]) can be combined with the measured data. This makes the analysis more accurate and this medical info is available for the supervisors.
5. **Supervisor:** Most of the health tracking apps just support self-management. However, if a supervisor could be notified when abnormalities are observed (e.g., in [22, 33]), it helps to control emergencies. The supervisor could be an organization, such as a hospital, or a person (e.g., the patient's family members and doctors).
6. **App Domain:** Health monitoring apps can be used for measuring general vital signs (e.g., in [33]) or be specialized for a specific disease (e.g., "Blood Pressure Diary").
7. **User Multiplicity:** The reviewed health tracking apps are used by one user (e.g., "Mi Fit") or they are used by multiple users (e.g., "Diabetes: M-Management & Blood Sugar Track App").

Figure 2 shows the resulting feature model. The importance of each feature is determined using a logical operator. The features that are marked with a solid circle (e.g., Functionality) are mandatory and must be supported by a health monitoring app. Features with an empty circle are optional (e.g., Storage). One or more sub-features that are connected by filled arcs can exist in the final app whereas an empty arc means that only one of the features should exist. Sub-features that are indicated with dashed boxes (e.g., iOS applications) are not supported by the current version of HealMA.

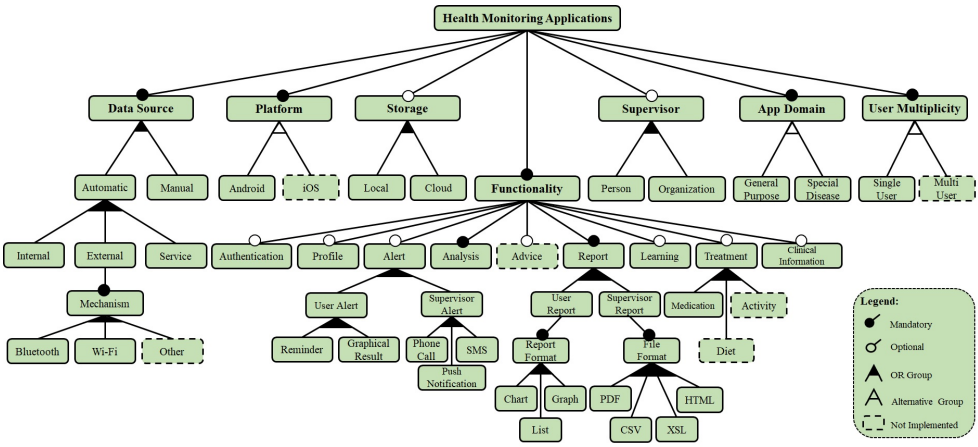

Fig. 2 Feature model of health monitoring domain

3.1.3 Design

We built our DSML based on the features extracted in the analysis phase. Figure 3 shows an overall view of HealMA metamodel, which consists of four main parts shown in different colors: 1) Participants (yellow); 2) Health concepts (blue); 3) Data providers (purple); 4) App functionalities (green). A more detailed view of the metamodel is presented in Figure 17 of Appendix A.

Participants refers to the roles of various participants in the health monitoring process, including patients (represented with the *Patient* class) whose health condition is monitored. The patient's personal information is represented by the *Profile* class and a patient's medical information is modeled using the class *Medical History*. We define the *Supervisor* role who is the participant who monitors the patient's health condition. The supervisor can be either a *person* such as a physician or an *organization*, for example, a health care clinic.

Health Concepts represent concepts related to a particular disease. *Parameter List* is a container that represents the different *Parameters* that are monitored. These include physical measurements such as blood pressure, heartbeat rate, and body temperature (represented using the *Physical* class) and

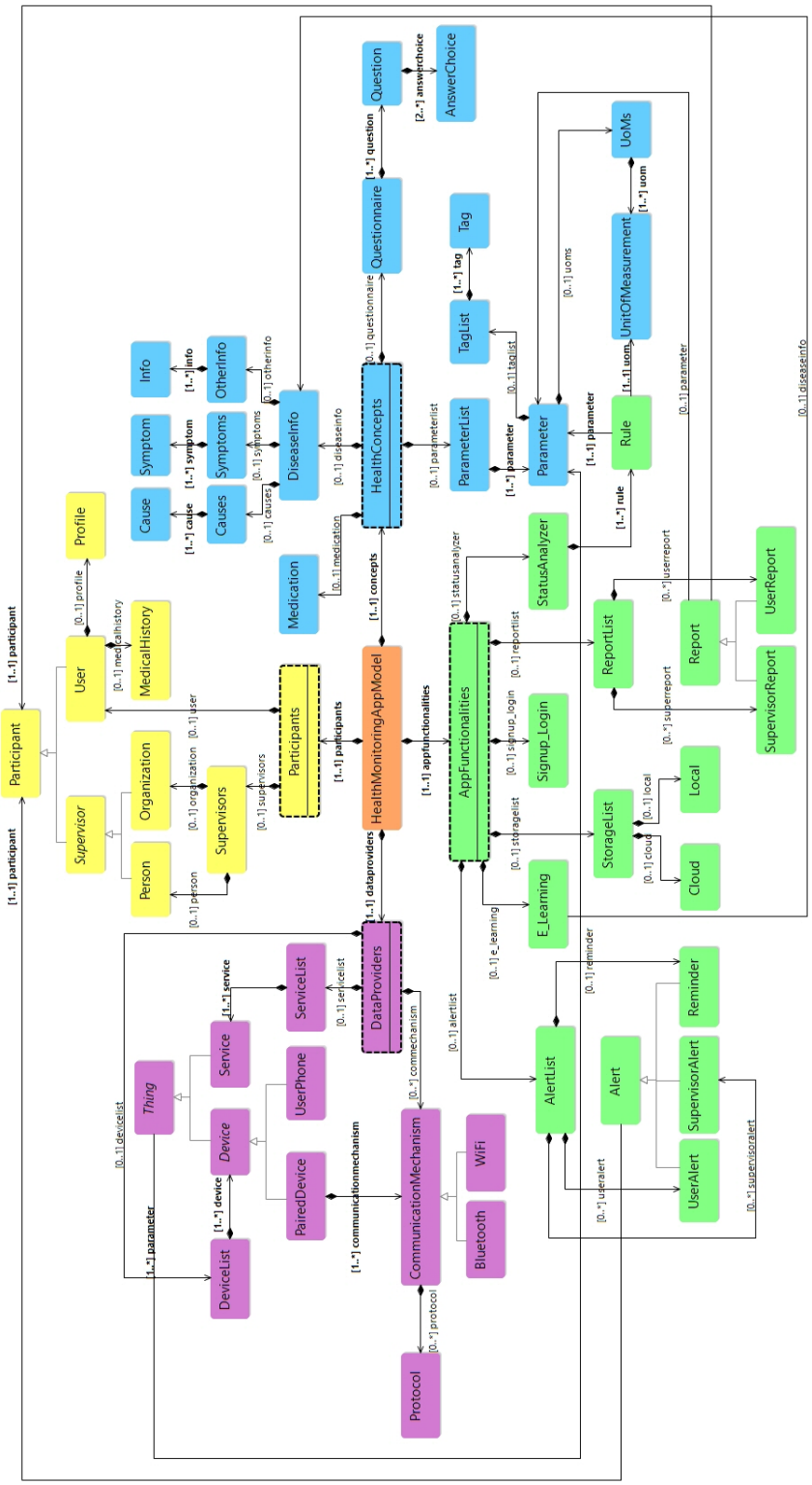

Fig. 3 Health monitoring app generation metamodel (in summary)

Environmental parameters such as humidity, temperature, and AQI. The geographical location of a patient is represented using the *Location* class. A patient's physical activities which are measured using step counts and acceleration are defined using the *Activity* class. Each parameter has an associated *Units of Measurement* and *Tags*. For example, the temperature can be measured either in Celsius or Fahrenheit, which are units of measurement. Tags capture contextual information about the measurement. For example, the temperature of a patient can be taken after rest time or after doing physical exercise, etc.

However, not all physical parameters are quantifiable. For example, a headache could be something that a patient should be able to report. For such health-related physical parameters, we use the *Questionnaire* class, which has an association with the class *Questions*. These classes are used to inquire about other health conditions that cannot be captured using sensors.

Specific information about the target disease could be defined as *Causes* and *Symptoms*. If there is information that is not a symptom or cause, it could be defined using *Other Info*. Adding the *Medication* concept helps keep track of the medication taken by the patient.

The *Data Providers* concept includes *Things* that are needed for data collection and communication management. *Things* could be either *Device* or *Service*. It means that the controlling parameters could be received from a number of devices and services. *Device* can be either the *patient's smartphone* or an *external device* such as a smartwatch that pairs with the patient's device using a *Communication mechanism*. In this work, we just support Bluetooth and Wi-Fi for communicating with external devices. However, it could be extended easily.

App Functionalities model the app's functionalities such as facilities for registering and logging into the app using the *Signup and Login* concept. The collected data can be stored in *Local* or *Cloud* storage by defining the concept of *Storage Spaces*. The *E-Learning* concept helps increase the patient's knowledge about the disease through specific information that is added in *Health* concept. The *Status Analyzer* concept is used to define some *Rules* for the simple processing of the patient's device to check parameters' normal thresholds. The *Report* concept represents the different types of reports such as charts and history lists. The concepts such as *User Alert*, *Reminder*, and *Supervisor Alert* are used to notify the patient and supervisors in case of emergency. For example, if the collected data for a parameter violates predefined thresholds, the supervisor must be notified by a message, and the patient must receive an alert from the app. Adding the reminder concept makes it possible to add reminders for the measurement.

3.1.4 Implementation

We designed our metamodel using the Eclipse Modeling Framework (EMF)[5]. EMF is a well-known open-source modeling framework and code generation

[5]Eclipse Modeling Framework(https://www.eclipse.org/modeling/emf/)

facility, used in building app generation tools [34]. This framework allows defining metamodels using the domain concepts and their relationships.

3.1.5 Deployment

The last phase is to make the DSML available for use by deploying it. For this, we created a modeling editor and defined a set of graphical notations. This modeling tool has to be installed in Eclipse IDE. Then, the developer could model the applications using the proposed graphical notations. The created models must conform to the metamodel. Figure 4 shows the concrete syntax of the language. More details on the editor are provided in Section 3.2.

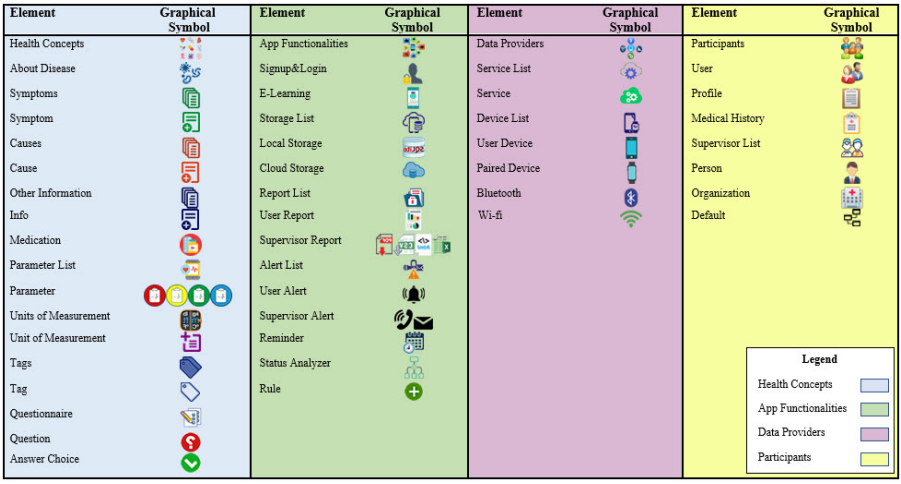

Fig. 4 The language concrete syntax

3.2 Graphical Modeling Tool

To create health monitoring app models in a user-friendly environment, we developed a graphical modeling editor using Sirius[6]. Sirius is an Eclipse-based tool that allows the creation of customized modeling editors. We selected Sirius because of its flexibility in supporting a wide range of customization when defining modeling editors. The modeling editor is added to Eclipse as a plugin to make it possible to easily create models by dragging and dropping the language elements to the modeling area. The resulting models are saved in an XMI file.

As shown in Figure 5, our modeling tool consists of three main parts: 1) Palette, 2) Canvas, and 3) Properties, as described in the following.

[6] https://www.eclipse.org/sirius/

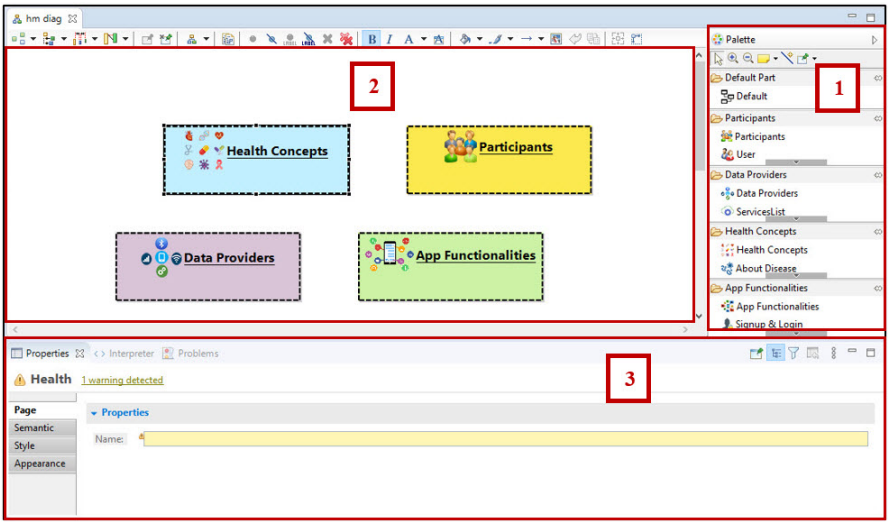

Fig. 5 Environment of HealMA modeling editor

1. **Palette:** This part provides the language elements categorized into four categories: Participants, Health Concepts, Data Providers, and App Functionalities. This palette helps to learn the language faster and makes modeling easier. In addition to these four categories, there is another category with an element named *Default* that enables the developer to create all the necessary parts of each model (mandatory features in the feature model) in the shortest time, automatically.
2. **Canvas:** To create a model, the developer adds the model elements to a canvas through a drag-and-drop mechanism. Then, the editor shows the graphical representation of the model to the developer. Each of the four main parts of the application model is indicated by the corresponding color that we used in explaining our metamodel (see Figure 3).
3. **Properties:** This part provides the attributes of each model element to be filled. After the developer adds model elements to the canvas, the required fields must be filled to complete the application model.

3.3 Constraints and Validation Rules

We defined many constraints and validation rules to prevent the creation of incomplete and invalid models. To this end, we added 38 AQL (Acceleo Query Language) rules to the modeling tool to catch potential faults. The faults are categorized into three levels of severity: information, warning, and error. Error messages are generated to prevent the user from creating the same elements which must be unique in the application model. Warning messages are generated to prevent important fields from being left blank. Information messages are generated to express the description needed to fill a field. As a result, during modeling, if any rule is violated, the editor shows an appropriate message

based on the severity level of the rule. Table 3 shows the number of rules defined for each level, while Table 4 shows the rules with the highest severity.

Table 3 The number of rules for different severity levels

Error	Warning	Information
9	25	4

Table 4 The rules with highest severity level which causes error

No.	Rule	Target Class	Usage reason
1	not(self.siblings())→collect(i|i.isdefaultuser) →includes(self.is default user)	User	To prevent defining more than one default user.
2	not(self.siblings())→collect(i|i.name) →includes(self.name)	Paired Device	To prevent defining the same devices.
3	not(self.siblings())→collect(i|i.name) →includes(self.name)	Parameter	To prevent defining the same parameters.
4	not(self.siblings())→collect(i|i.value) →includes(self.value)	Unit of Measurement	To prevent defining the same units of measurement for a parameter.
5	not(self.siblings())→collect(i|i.value) →includes(self.value)	Tag	To prevent defining the same tags for a parameter.
6	not(self.siblings())→collect(i|i.value) →includes(self.value)	Cause	To prevent defining the same causes.
7	not(self.siblings())→collect(i|i.value) →includes(self.value)	Symptom	To prevent defining the same symptoms.
8	not(self.siblings())→collect(i|i.title) →includes(self.title)	Info	To prevent defining the same topics of information.
9	not(self.siblings())→collect(i|i.question) →includes(self.question)	Question	To prevent defining the same questions.

For example, for each instance of the class Parameter, we can issue an error message when the developer creates a parameter that already exists. Figure 6 shows different steps to define this error message and the result of validating a model against this rule. Figure 6(a) shows the required AQL query to prevent defining the same parameters. Figure 6(b) shows the customization of the message level and message text. Figure 6(c) shows an error message when the developer tries to define duplicate parameters. In addition to the error message, we can show an 'information' message to help a developer choose the type of parameters and 'warn' the developer when he/she forgets to enter the required fields. These messages are defined following the same process as the error message is defined.

3.4 Model-to-Code Transformation Engine

To generate the code from the model, we developed a model-to-code engine. We used Acceleo model-to-text language [35], which is a template-based code

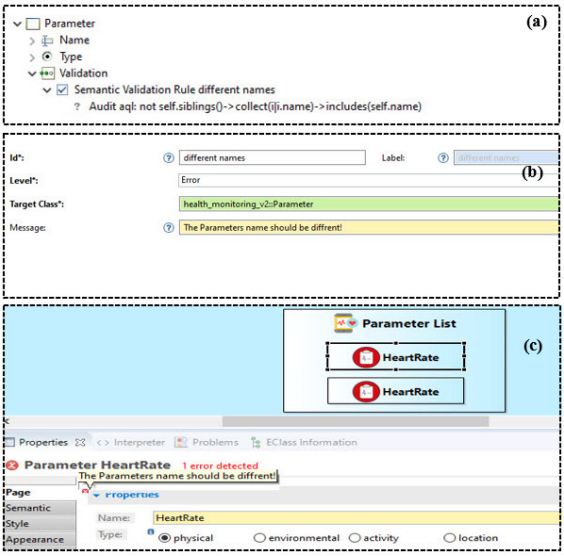

Fig. 6 Process of defining and showing an error message to prevent creating duplicate parameters

generator tool, to create the engine. The engine includes several transformation rules that take the model as an XMI file and returns the required files for the Android app as output. The app logic code is generated in Java, and the user interface is generated in XML. In addition, the transformation rules generate comments to help developers fill some parts manually, fix errors, and complete the Android project. Listing 1 shows a code excerpt that is written in Acceleo. This template generates the required Java method to insert measured data into the corresponding table in the local database.

```
1        [template public InsertParam(aParam : Parameter)]
2        public void add_[aParam.name/](Parameter_DataModel params){
3
4        SQLiteDatabase db=this.getWritableDatabase();
5        ContentValues values=new ContentValues();
6
7        values.put(Constants.[aParam.name.toUpper()/]_VALUE,
8        params.get[aParam.name.toUpperFirst()/]());
9        values.put(Constants.[aParam.name.toUpper()/]_DATE,
10       params.get[aParam.name.toUpperFirst()/]_date());
11
12       db.insert(Constants.TABLE_NAME_[aParam.name.toUpper()/], null,
             values);
13       Log.i("tag", "[aParam.name/] is added successfully....");
14       db.close();
15       }
16       [/template]
```

Listing 1 Code excerpt of an Acceleo template to generate Java method to add data to parameter table

3.5 Architecture of HealMA-Generated Apps

The health monitoring apps generated by HealMA follow the Model-View-Controller (MVC) architecture pattern. As it can be seen in Figure 7 based on the MVC architecture each app is divided into three main layers: 1) Model 2) View 3) Controller.

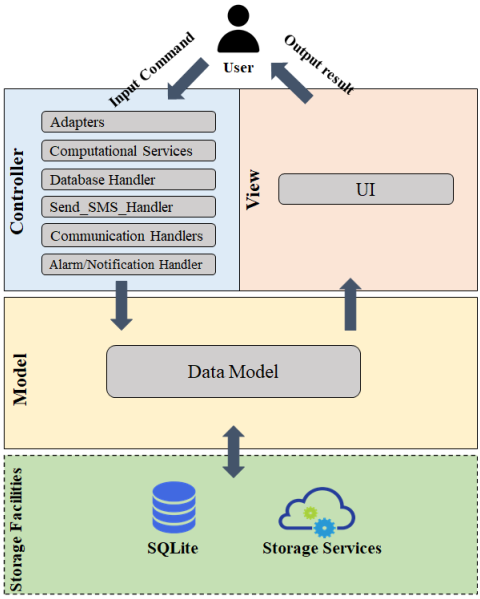

Fig. 7 Architecture of health monitoring applications generated by HealMA

- **Model** layer is responsible for storing/retrieving data in/from database and cloud storage services using object classes.
- **View** layer is responsible for visualizing the appropriate user interface to the user based on the user command.
- **Controller** layer is responsible for getting the user commands and updating the model layer based on the command. It is also responsible for application logic.

In an Android project, activities are used for both the view and controller layers. To be more precise, the view layer consists of a combination of XML layout resources and the parts of Java activities files that update the user interface. However, the parts of the activities that update the object classes and handle the logic of the program are related to the controller layer. The Java classes from which objects are built are in the model layer [36].

3.6 Development Process using the Proposed Framework

Figure 8 shows a detailed process that developers should follow to generate an Android health monitoring app using HealMA. This process consists of the following steps.

1. In the Requirement Gathering step, the developer should determine the requirements of the application. However, HealMA does not provide any facility for doing this step. Therefore, this step is considered a prerequisite step.
2. The next step is Modeling, in which the developer should model the application using the modeling editor. The model elements can be added in three ways. They can be added by utilizing the default parts, they can be designed from scratch, or they can be created by the combination of these two approaches.
3. In the Automatic Code Generation step, the app code (including Java and XML files) is generated automatically by the model-to-code transformation engine.
4. Finally, in the Code Customization step, the developer has to transfer the generated code into the Android Studio IDE, complete the project according to the comments, and customize the project as needed.

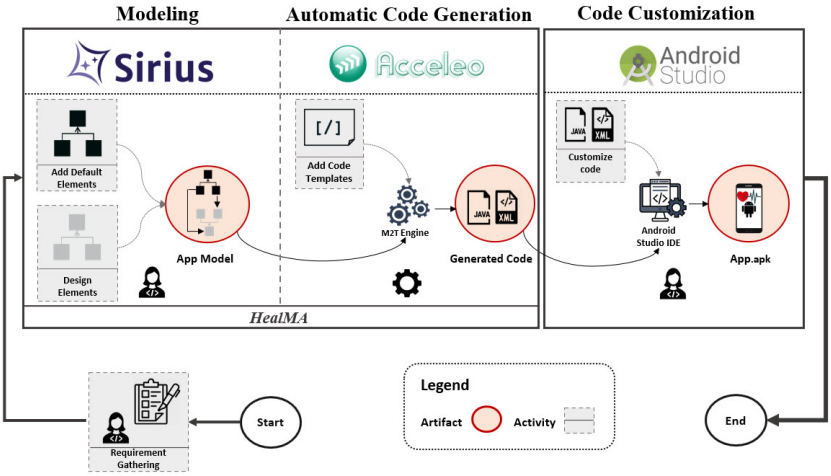

Fig. 8 Development process of health monitoring applications using HealMA

4 Evaluation

In this section, we evaluate the HealMA framework using three criteria: 1) Applicability, 2) Productivity, and 3) Usability. More precisely, we answer the following research questions:

- RQ1 (Applicability): Can we use HealMA to generate different Android-based health monitoring applications?
- RQ2 (Productivity): Does HealMA increase the productivity of developers?
- RQ3 (Usability): How usable are the apps generated by HealMA?

To answer the first two questions, we adapted the guidelines introduced by Runeson and Host [37] to the current research, and implemented four case studies. The detail of this part of the evaluation is explained in Section 4.1. To answer the third research question, we conducted a user study to assess the usability of one of the generated apps (COVID-19 monitoring app) from the users' perspective, following the guidelines proposed by Wohlin et al. [38]. Our usability measures are based on the ones proposed by Hussain et al. [39] and Gharaat et al. [40]. These measures are then structured using the goal question metric (GQM) approach [41]. The detail of this part of the evaluation is explained in Section 4.2.

4.1 Case studies

In the following sections, we address the first two research questions: RQ1 and RQ2.

4.1.1 RQ1 (Applicability): Can we use HealMA to generate different Android-based health monitoring applications?

To answer this question, we present four case studies in which we use HealMA to automatically generate four different monitoring apps for the following diseases: 1) Chronic Obstructive Pulmonary Disease (COPD), 2) High Blood Pressure, 3) COVID-19, and 4) Diabetes. To show how HealMA can be used to generate a monitoring app, we explain the details of the development process for the first case study. The same approach was followed for the other three case studies.

COPD Monitoring

COPD is a chronic disease that makes breathing difficult [42]. Despite its progressive nature, it could be managed and controlled. Early diagnosis, continuous monitoring, and reducing risk factors can help patients control COPD [43]. This disease can be caused by tobacco smoking, indoor and outdoor air pollution, occupational dust, respiratory infections, chemicals, and so on. The symptoms can be dyspnea, chronic cough, shortness of breath, sputum production, wheezing, and chest tightness [43, 44]. A set of environmental parameters such as temperature, humidity, and rising levels of some pollutants (e.g., CO, PM2.5, and PM10) exacerbate COPD. Such exacerbation can be detected by monitoring physical parameters such as heartbeat rate, blood pressure, body temperature, respiration rate, and blood oxygen level [25, 45, 46]. These parameters should be monitored continuously in the case of COPD.

Based on the development process described in Section 3.6, to generate a COPD monitoring mobile app by HealMA, we follow the following steps.

Step 1

In the first step, we defined the requirements for the COPD monitoring app. Figure 9 shows the requirements of this app as a use case diagram.

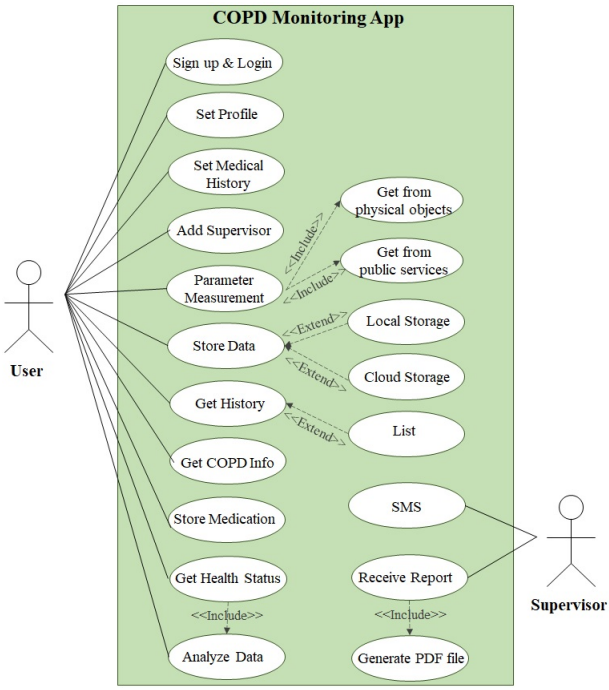

Fig. 9 Requirements of the COPD monitoring app

Step 2

We used the HealMA modeling editor to design the COPD model. Then, we applied the validation rules to correct and complete the model. Figure 10 shows the application model for the COPD monitoring app.

Step 3

After the model is completed, with the help of the transformation engine, the XML and Java files of the application are generated automatically.

Figure 11 shows the generated packages based on the MVC architecture. The *controller* package consists of several sub-packages. The *adapter* package consists of adapter classes that act as a bridge between the model layer and the view layer. The *alarm notification handler* consists of classes that make it possible to add the alarm and notification in the app. The *communication handler* is responsible for handling the network communications. The *Bluetooth/Wi-fi handler* is used to receive health data from external devices. The *public services*

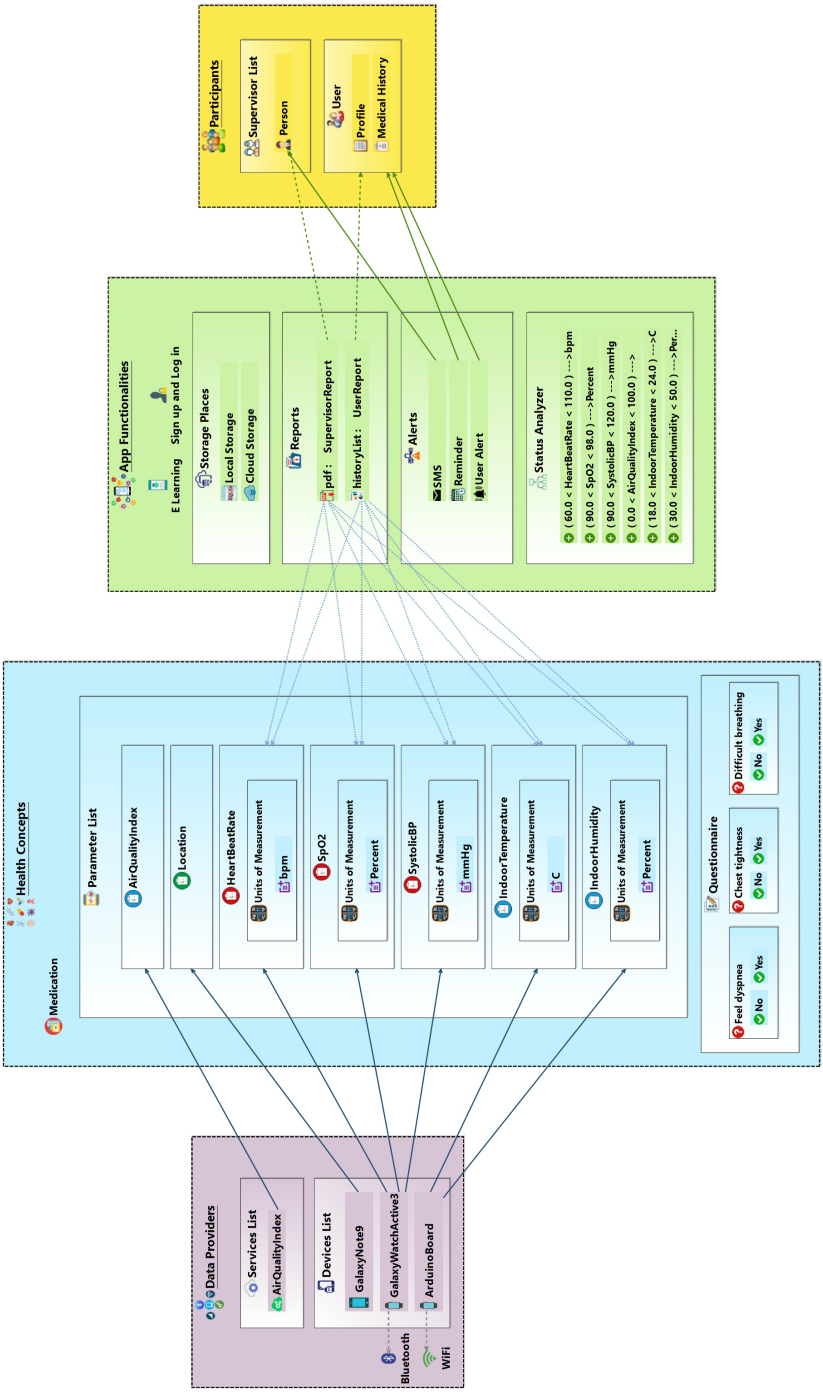

Fig. 10 COPD application model

handlers contain classes to receive data from public services such as services providing air quality index. The *server storage handler* consists of classes that are responsible for communicating with the storage cloud server. The *computational services* package implements services for mathematical and logical operations to check the patient health status. The *database handler* consists of the classes that are responsible for storing/retrieving on/from the SQLite database. Finally, the *send SMS handler* package contains a class to send SMS to observers.

The *model* package consists of Java classes to create the objects for handling the logic of the program.

The *presentation* package consists of activities and layouts. The *Activities* are Java classes that are responsible for both updating the user interface and some tasks which are related to the controller layer. The *layouts* package are XML layout resources for the user interface.

There is one more package that includes the required manifest file of the project.

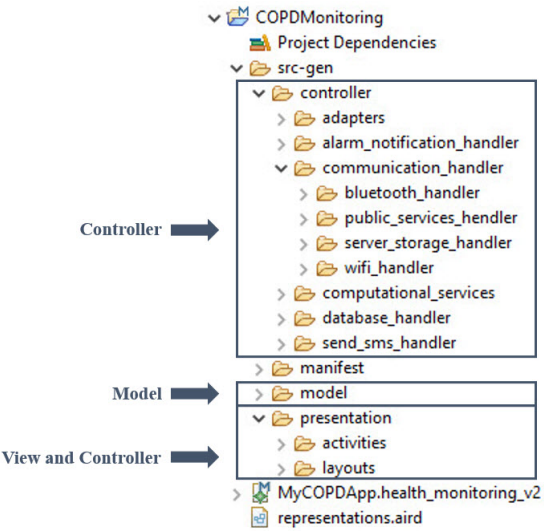

Fig. 11 Generated packages for the COPD monitoring application

step 4

We opened the generated code as an Android project in Android Studio. Then, we completed the project by addressing the comments that are generated by HealMA. Finally, we generated the APK file for the COPD monitoring app. The COPD monitoring app APK file is available at this address[7].

Figure 12 shows different pages of the generated COPD monitoring app. Figure 12(a) shows three main parts of the app including the signup and

[7] https://mdse.ui.ac.ir/Tools/HealMA/COPDMon.apk

login page, dashboard, and drawer. After the user signs up and logs into the app, he/she should use the drawer to fill out the profile, medical history, and supervisor information. Then, he/she can use other parts of the app. Figure 12(b) shows the measurement page, which is the most important part of the app. After the user device is paired to the external devices, it can receive the measured data, check the thresholds, and return the result to the user. To test the storage and retrieval of the data on the cloud, we developed a local PHP server. Also, to test Bluetooth and Wi-Fi communications, we developed an Android app that sends data to the generated COPD monitoring application via Bluetooth and Wi-Fi. Then, as depicted in Figure 12(c), the stored data can be shown to the user and can also be sent to the supervisor as a report. The user can obtain information about COPD and improve his/her knowledge as shown in Figure 12(d), and can also define his/her medications in pages depicted in Figure 12(e). Additionally, the user has the option to set a reminder for the measurement time or other important times in the reminder part which is shown in Figure 12(f).

Blood Pressure Monitoring

Blood pressure is measured as the blood circulates from the heart to other parts of the body through the vessels. Blood pressure rises and falls during the day. However, if it stays too high for a long time, it is called hypertension and raises the risk of heart, brain, and kidney problems. This condition is the main reason for premature death worldwide [47, 48]. Recently, many applications have been developed to monitor hypertension. We used existing apps to collect requirements about blood pressure monitoring. More specially, we chose three most popular blood pressure monitoring apps from Google Play as a reference, which are shown in Table 5 along with their number of downloads and ratings. We aggregated the functionalities supported by these apps and added other requirements such as automatic data recording and alerts to come up with a full set of features for the HealMA-generated app.

Following the same process as for the COPD app, we generated the blood pressure app simply by dragging and dropping the relevant model elements using the HealMA modeling editor. Figure 13 shows the main screenshots of the HealMA-generated Blood Pressure monitoring app. The app APK file is available at this address[8].

COVID-19 Monitoring

COVID-19 is a new infectious disease that mainly causes respiratory problems. Fever, dry cough, and tiredness are the most common symptoms of COVID-19, and based on the latest research, factors such as age, gender, heartbeat rate, respiratory rate, oxygen saturation, and chronic kidney problems affect the COVID-19 mortality rate [49, 50]. However, due to the infectious nature of COVID-19, direct nursing of patients and controlling such factors is dangerous for the patient's family members and the medical staff. Remote monitoring not

[8] https://mdse.ui.ac.ir/Tools/HealMA/BloodPressureMon.apk

Fig. 12 The screenshots of COPD monitoring app: a) main pages b) measurements c) reports d) learning about disease e) medication storage f) reminder

only helps supervisors to be safer but also provides stored data for researchers to study the disease. Due to the fact that at the time of writing this paper, there is no well-known COVID monitoring app available on the Google play store, we relied on the research articles to extract the features and functionalities of such an app.

Figure 14 shows some screenshots of the COVID-19 monitoring app generated using HealMA. It should be noted that this app is generated following

Table 5 Functionalities for blood pressure monitoring app

App name		Blood Pressure	Blood pressure Tracker & bp diary	Blood Pressure Tracker	Our App
Downloads		10M+	1M+	500k+	-
App functions↓ Rate		4.2	4.1	4.6	-
Profile		○	○	○	●
Data recording	Manual	●	●	●	●
	Automatic	○	○	○	●
Storage place	Local	●	●	●	●
	Cloud	●	○	●	●
Data visualization	History list	●	●	●	●
	Charts	●	●	●	●
Supervisor notification	Report	●	○	●	●
	Alert	○	○	○	●
Reminder		●	○	○	●
Learning about disease		○	●	○	●

Supported ● Not Supported ○

the same process as the previous case studies. The app APK file is available at this address[9].

Diabetes Monitoring

Insulin is a hormone that moves the sugar (glucose) from the blood to body cells to be used as energy. Diabetes occurs when the body is unable to produce or use insulin properly [51]. There are three types of diabetes, type 1, type 2, and gestational. However, type 2 diabetes is the most common. Continuous controlling of some physical parameters such as blood glucose, blood pressure, and cholesterol besides improving the lifestyle by quitting smoking, controlling the weight, and healthy eating help to manage type 2 diabetes [52]. Health monitoring apps can be helpful by providing facilities for the mentioned guidelines. To develop an application to help people with type 2 diabetes, we selected three diabetes tracking apps with a high number of downloads from the Google play store, and considered their main functionalities as the requirements of our application. Table 6 shows the apps and their number of downloads, ratings, features, and the features we considered for our app. Figure 15 shows some screenshots of the Diabetes monitoring app generated using HealMA. The app APK file is available at this address[10].

We believe that the four apps that were generated using the HealMA framework demonstrate the applicability of HealMA to generate different health monitoring apps for various domains. This results in the 'yes' answer to our first research question, RQ1.

[9] https://mdse.ui.ac.ir/Tools/HealMA/COVIDMon.apk
[10] https://mdse.ui.ac.ir/Tools/HealMA/DiabetesMon.apk

Fig. 13 The screenshots of Blood Pressure monitoring app

4.1.2 RQ2 (Productivity): Does HealMA increase the productivity of developers?

To answer the second research question, we compared the "number of automatically generated lines of code" with the "manually added lines" for each case study. Table 7 shows the numbers, and Figure 16 shows the percentage of these numbers. As we can see, in all cases, more than 90% of the code is automatically generated, and this shows that HealMA helps developers build the apps in a more efficient manner.

In addition, to compare the development time when using HealMA compared to creating the apps from scratch (i.e., following a pure code-centric approach), we used the heuristic presented by Kung [53] which claims that an average developer can write 100 lines of code (LoC) per day. Based on this heuristic, we compared the time it takes to develop each of the apps from scratch, with the time of developing the same apps using HealMA. As it is shown in Equation 1, to obtain the time for the code-centric method, the total number of LoC for each project should be divided by 100. However, when HealMA is used for development, as it is shown in Equation 2 the modeling time will be added to the time needed for manual LoC added to complete the project. Table 8 shows the results.

$$Time(day) = \frac{LoC(Total)}{100} \tag{1}$$

Fig. 14 The screenshots of COVID-19 monitoring app

$$Time(day) = ModelingTime + \frac{LoC(ManuallyAddedCode)}{100} \tag{2}$$

To summarize, as depicted in Figure 16, at least 92% of XML code and 95% of Java code is generated automatically for each case study. Based on these results, we conjecture that HealMA increases productivity during development, which results in the 'yes' answer to RQ2.

4.2 RQ3 (Usability): How usable are the apps generated by HealMA?

To evaluate the usability of the apps generated by HealMA, we selected one of the apps that was generated using our framework (COVID-19 Monitoring app), and conducted a user study following the guidelines provided by Wohlin et. al [38], ISO 9241-11 quality model [54], and measures proposed by Hussain et al. [39] and Gharaat et al. [40]. The user study consists of three steps: 1) Scoping and planning, 2) Operation, and 3) Discussion. In the first step, we defined the goal, set up the experiment, and chose the participants. In the second step, we conducted an online questionnaire with the participants. Then, in the third step, we discussed the results achieved from the questionnaire. These three steps are explained in more detail in the following sections.

Table 6 Functionalities for diabetes monitoring app

App name		Diabetes Diary-Blood Glucose Tracker	Blood Sugar Log-Diabetes Tracker	Blood Sugar Tracker: Diabetes Test Glucose Logger	Our App
Downloads		100k+	100k+	100k+	-
App functions↓ Rate		4.6	4.5	4.6	-
Sign up & Log in		○	○	○	●
profile		○	○	●	●
Medical history		○	○	○	●
Data recording	Manual	●	●	●	●
Storage place	Local	●	●	●	●
Data visualization	History list	○	○	●	●
	Charts	●	●	●	●
Supervisor notifications	Report	○	●	○	●
	Alert	○	○	○	●
Reminder		●	●	●	●
Learning about disease		○	○	○	●
Medication		●	●	○	●

Supported ● Not Supported ○

Table 7 Automatically generated code vs. manually added code for each case study

	Automatically generated code		Manually added code		
Case study	Java	XML	Java	XML	total
COPD	6438	3107	350	272	10167
Blood Pressure	2110	1261	58	13	3442
COVID-19	2699	1584	43	24	4350
Diabetes	4838	2291	205	186	7520

4.2.1 Scoping and Planning

According to ISO 9241-11 [54], usability is considered as user-friendliness and ease of use for the users. Therefore, end-users (COVID-19 patients) should be satisfied with the generated app. In order to define the goals, we followed the GQM approach [41]. This approach helps find the factors that may have an effect on usability and define a set of questions to make usability more quantifiable. We adapted the goals defined by Hussain et al. [39], by focusing on simplicity, accuracy, time taken, and attractiveness as the goals of the study. Then we set up a questionnaire with 12 questions to evaluate these goals. The answer to each question has to be selected amongst these options: 'very little', 'little', 'moderate', 'much', and 'very much'. Then, to evaluate the goals, we asked 20 participants to answer the questionnaire. The participants have been selected from people who experienced COVID-19, either themselves or their close family members. They were of different age groups from 15 to 50 and had different education levels.

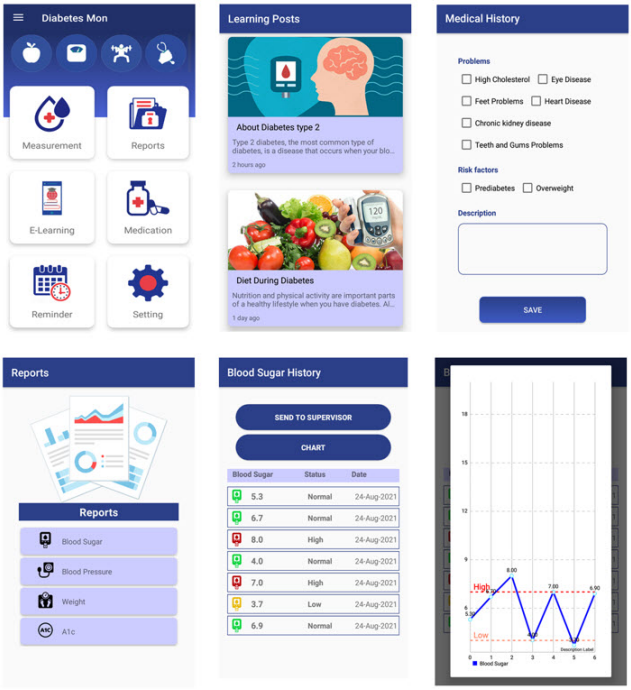

Fig. 15 The screenshots of Diabetes monitoring app

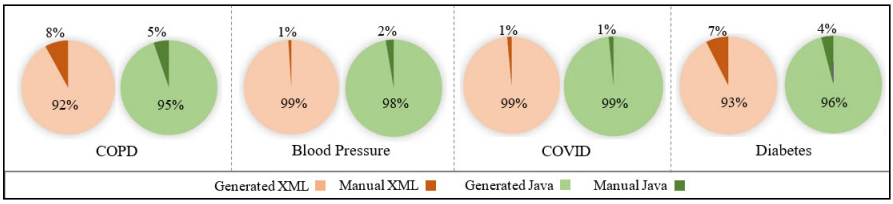

Fig. 16 The percentage of automatically generated code vs. manually added code for four case studies

4.2.2 Operation

The experiment was conducted online because of the current pandemic. We provided the COVID-19 monitoring app to participants, which is generated as the third case study. Then, we asked the participants to install the app and use it. After that, we asked them to fill out the online questionnaire to evaluate the usability of the app. Table 9 shows the questions and the extracted results from the answers.

4.2.3 Discussion

As mentioned earlier, the answers to each question vary from 'very little' to 'very much'. If the 'very much' and 'much' answers are considered as the users' satisfaction, in general, users' satisfaction varied between 70% to 100% for each question. To achieve user satisfaction for each goal, i.e, Simplicity, Accuracy, Time taken, and

Table 8 Comparison between the time of automatic implementation by HealMA and manual implementation for each case study

Case study	Manual implementation (Days)	Automatic implementation by HealMA	
		Modeling (Minutes)	Completing & Customization (Days)
COPD	101	120	∼7
Blood pressure	34	30	∼1
COVID-19	43	40	∼1
Diabetes	75	60	∼4

Attractiveness, we averaged the user satisfaction for the questions of that goal. Table 10 shows the general user satisfaction for the goals. It can be seen that 96.6% of users believed that working with the COVID-19 monitoring app is easy and 88% of them considered it Accurate. 95% were satisfied with the response time, and 91.6% of them believed that the COVID-19 monitoring app is attractive.

The results show that the COVID-19 monitoring app is considered usable by the users, which answers RQ3.

4.3 Threats to Validity

In this section, based on Runeson and Host [37] we describe the threats to validity.

Construct validity

Construct threats refer to differences between theoretical assumptions and the results observed. We evaluated the framework from different perspectives based on well-known methods. To evaluate the applicability of HealMA, we implemented four case studies with different requirements. We measured the development time for each case study to measure productivity. The results for measuring productivity may be different by changing the implemented cases.

Internal validity

Internal threats refer to factors that affect the evaluation and change the results. Automatic code generation reduces the need for programming skills and development time. However, different programming styles may increase the time to customize the code, which in turn can affect the development time. To mitigate this threat, an effort was made to generate code that is easy to understand with added comments to help developers complete the project.

External validity

External threats refer to factors that prevent generalizing the findings and results. To mitigate this threat, we generated four different apps that cover various health monitoring aspects.

Reliability validity

Reliability threats refer to the dependence of the results on the researchers. In this research, for the evaluation of applicability, we analyzed the health monitoring app

Table 9 Questionnaire results

			Answer				
Measure	Goal	Question	1	2	3	4	5
	Simplicity	How simple is the installation process of the App?	0%	0%	5%	5%	90%
		How simple is it to learn the App functionalities?	0%	0%	0%	50%	50%
		How simple is it to use the App?	0%	0%	5%	50%	45%
Effectiveness	Accuracy	How accurate are the results provided by the App about health status?	0%	0%	5%	50%	45%
		Is the App able to prevent errors and guide users with appropriate messages?	0%	15%	15%	35%	35%
		Is the App able to complete given tasks successfully?	0%	0%	0%	50%	50%
Efficiency	Time Taken	Does the App respond in a reasonable time?	0%	0%	0%	80%	20%
		Does the App complete the given tasks in a reasonable time?	0%	0%	5%	5%	90%
		Does the App take a reasonable time for monitoring the user's health status?	0%	0%	10%	45%	45%
Satisfaction	Attractiveness	How pleasant is the UI of the App?	0%	0%	10%	40%	50%
		How satisfied is the user with colors, font size, and icons in the App?	0%	0%	5%	20%	75%
		Does the user enjoy working with the UI of the App?	0%	0%	10%	30%	60%

1=Very Little, 2=Little, 3=Moderate, 4=Much, 5=Very Much

Table 10 User satisfaction for each goal

Goal	Satisfaction percent
Simplicity	96.6%
Accuracy	88%
Time Taken	95%
Attractiveness	91.6%

domain by reviewing several applications and academic studies and designed the language based on the common features. However, we may have missed some studies and apps. Therefore, all the features in health monitoring applications may not be implemented in the current version of HealMA. On the other hand, for the evaluation of usability, the sample size is not large enough. To mitigate this threat, we chose people from different age groups with various education levels.

4.4 Limitations

In this section, we discuss the potential limitations of the HealMA framework and the apps generated using HealMA.

Limitations of HealMA

As we stated in Section 3.1.2, we relied on the analysis of several apps and academic research work and extract the main concepts of the remote health monitoring domain. These concepts form the foundations on which HealMA metamodel is built. However, we can not claim that HealMA can generate any health monitoring mobile app. It is only limited to the generation of apps with the features defined in the HealMA metamodel. To support additional concepts, we need to extend the HealMA metamodel, which will result in changing other parts of the framework, i.e., the modeling editor, validation rules, and code generator. This said, we believe that the current version of HealMA covers a wide range of standard features for creating useful health monitoring apps.

Limitations of the apps generated by HealMA

The monitoring apps generated by the current version of HealMA have the same UI design, and there is no support for UI customization in the modeling step. The only way to modify the UI is to change the generated code itself, which may create inconsistencies with the models created using HealMA DSML. To support UI customization at the modeling level, we need to improve the DSML. This will require a good understanding of UI design concepts.

5 Conclusion and Future work

In this paper, we presented HealMA, a model-driven engineering framework for generating Android-based remote health monitoring apps. HealMA consists of a domain-specific modeling language, a modeling editor, validation constraints, and a model-to-code transformation engine, all packaged in an Eclipse plugin. The modeling language includes the main concepts of the IoT-based health monitoring apps as well as the relationships between them. The modeling editor makes modeling easier. It includes some predefined AQL rules to avoid creating wrong and incomplete models. The model-to-code transformation engine takes the designed model of the app as input and returns the generated XML and Java code of the app as output, which then is imported to the corresponding Android project for generating the APK of the app.

We evaluated HealMA from three different perspectives: applicability, productivity, and usability. To show that HealMA is applicable, we implemented four different case studies. Then, we compared the development time when we used HealMA to a code-centric approach. We also evaluated the usability of HealMA-generated apps by conducting an experiment and asking 20 participants to provide their feedback through a questionnaire. The results of the evaluations showed both the applicability and usefulness of the HealMA framework.

In the future, we plan to extend HealMA with a focus on a wider range of diseases, communication methods, protocols, and standards. We also intend to add a set of transformations to our engine to generate iOS applications too. Then, we want to conduct more studies to assess the effectiveness and usability of HealMA.

References

[1] Sundaravadivel, P., Kougianos, E., Mohanty, S.P., Ganapathiraju, M.K.: Everything you wanted to know about smart health care: Evaluating the different technologies and components of the Internet of Things for better health. IEEE Consumer Electronics Magazine **7**(1), 18–28 (2017)

[2] World Health Organization Official Website: Integrated chronic disease prevention and control. [Online]. Available: https://www.who.int/. Accessed: 27 July, 2020

[3] Dhanvijay, M.M., Patil, S.C.: Internet of things: A survey of enabling technologies in healthcare and its applications. Computer Networks **153**, 113–131 (2019)

[4] Islam, S.R., Kwak, D., Kabir, M.H., Hossain, M., Kwak, K.S.: The Internet of Things for health care: a comprehensive survey. IEEE Access **3**, 678–708 (2015)

[5] Statista Official Website: Smartphone users worldwide. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. Accessed: 27 July, 2020

[6] Grand View Research Official Website: mHealth Apps Market Size. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/mhealth-app-market. Accessed: 2 July, 2021

[7] Inupakutika, D., Kaghyan, S., Akopian, D., Chalela, P., Ramirez, A.G.: Facilitating the development of cross-platform mhealth applications for chronic supportive care and a case study. Journal of Biomedical Informatics **105** (2020)

[8] Banos, O., Garcia, R., Holgado-Terriza, J.A., Damas, M., Pomares, H., Rojas, I., Saez, A., Villalonga, C.: mhealthdroid: a novel framework for agile development of mobile health applications. In: International Workshop on Ambient Assisted Living, pp. 91–98 (2014). Springer

[9] Qi, J., Yang, P., Min, G., Amft, O., Dong, F., Xu, L.: Advanced internet of things for personalised healthcare systems: A survey. Pervasive and Mobile Computing **41**, 132–149 (2017)

[10] Statcounter Official Website: Android market share worldwide. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide. Accessed: 27 May, 2021

[11] Brambilla, M., Cabot, J., Wimmer, M.: Model-driven Software Engineering in Practice, 2nd edn. Morgan & Claypool, MA (2017)

[12] Van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: An annotated bibliography. ACM Sigplan Notices **35**(6), 26–36 (2000)

[13] Panahandeh, M., Hamdaqa, M., Zamani, B., Hamou-Lhadj, A.: MUPPIT: A method for using proper patterns in model transformations. Springer Journal on Software & Systems Modeling (SoSym) **20**, 1491–1523 (2021)

[14] Pathinarupothi, R.K., Durga, P., Rangan, E.S.: IoT-based smart edge for global health: Remote monitoring with severity detection and alerts transmission. IEEE Internet of Things Journal **6**(2), 2449–2462 (2018)

[15] Wan, J., Al-awlaqi, M.A., Li, M., OGrady, M., Gu, X., Wang, J., Cao, N.: Wearable IoT enabled real-time health monitoring system. EURASIP Journal on Wireless Communications and Networking **2018**(1), 1–10 (2018)

[16] Al-khafajiy, M., Baker, T., Chalmers, C., Asim, M., Kolivand, H., Fahim, M., Waraich, A.: Remote health monitoring of elderly through wearable sensors. Multimedia Tools and Applications **78**(17), 681–706 (2019)

[17] Bauer, M., Bui, N., De Loof, J., Magerkurth, C., Nettsträter, A., Stefa, J., Walewski, J.W.: IoT reference model. In: Enabling Things to Talk, pp. 113–162. Springer, ??? (2013)

[18] Patel, P., Pathak, A., Teixeira, T., Issarny, V.: Towards application development for the Internet of Things. In: Proceedings of the 8th Middleware Doctoral Symposium, pp. 1–6 (2011)

[19] Bermudez-Edo, M., Elsaleh, T., Barnaghi, P., Taylor, K.: IoT-lite: a lightweight semantic model for the Internet of Things. In: 2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress, pp. 90–97 (2016)

[20] Einarsson, A.F., Patreksson, P., Hamdaqa, M., Hamou-Lhadj, A.: SmarthomeML: Towards a domain-specific modeling language for creating smart home applications. In: 2017 IEEE International Congress on Internet of Things, pp. 82–88 (2017)

[21] Hussein, M., Li, S., Radermacher, A.: Model-driven development of adaptive IoT systems. In: MODELS, pp. 17–23 (2017)

[22] Rhayem, A., Mhiri, M.B.A., Salah, M.B., Gargouri, F.: Ontology-based system for patient monitoring with connected objects. Procedia Computer Science **112**, 683–692 (2017)

[23] Gomez, J., Oviedo, B., Zhuma, E.: Patient monitoring system based on Internet of Things. Procedia Computer Science **83**, 90–97 (2016)

[24] Chellouche, S.A., Chalouf, M.A., Lemlouma, T.: Ontology-based pervasive m2m healthcare environment. In: 2013 First International Symposium on Future Information and Communication Technologies for Ubiquitous HealthCare, pp. 1–5 (2013)

[25] Ajami, H., Mcheick, H.: Ontology-based model to support ubiquitous healthcare systems for COPD patients. Electronics **7**(12), 371–400 (2018)

[26] Brambilla, M., Mauri, A., Umuhoza, E.: Extending the interaction flow modeling language (ifml) for model driven development of mobile applications front end. In: International Conference on Mobile Web and Information Systems, pp. 176–191 (2014)

[27] Usman, M., Iqbal, M.Z., Khan, M.U.: A product-line model-driven engineering approach for generating feature-based mobile applications. Journal of Systems and Software **123**, 1–32 (2017)

[28] Vaupel, S., Taentzer, G., Gerlach, R., Guckert, M.: Model-driven development of mobile applications for Android and iOS supporting role-based app variability. Software and Systems Modeling **17**(1), 35–63 (2018)

[29] Gharaat, M., Sharbaf, M., Zamani, B., Hamou-Lhadj, A.: Alba: a model-driven framework for the automatic generation of android location-based apps. Automated Software Engineering **28**(1), 2 (2021)

[30] Núñez, M., Bonhaure, D., González, M., Cernuzzi, L.: A model-driven approach for the development of native mobile applications focusing on the data layer. Journal of Systems and Software **161** (2020)

[31] Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Computing Surveys (CSUR) **37**(4), 316–344 (2005)

[32] Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, https://apps.dtic.mil/dtic/tr/fulltext/u2/a235785.pdf (1990)

[33] Swaroop, K.N., Chandu, K., Gorrepotu, R., Deb, S.: A health monitoring system for vital signs using IoT. Internet of Things **5**, 116–129 (2019)

[34] Eclipse Official Website: EMF. [Online]. Available: https://www.eclipse.org/modeling/emf/. Accessed: 3 July, 2021

[35] Eclipse Official Website: Acceleo Language. [Online]. Available: https://www.eclipse.org/acceleo. Accessed: 2 July, 2021

[36] Lou, T., et al.: A comparison of android native app architecture mvc, mvp and mvvm. Eindhoven University of Technology (2016)

[37] Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14**(2), 131–164 (2009)

[38] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, MA (2012)

[39] Hussain, A., Hashim, N.L., Nordin, N., Tahir, H.M.: A metric-based evaluation model for applications on mobile phones. Journal of Information and Communication Technology **12**, 55–71 (2013)

[40] Gharaat, M., Sharbaf, M., Zamani, B., Hamou-Lhadj, A.: Alba: a model-driven framework for the automatic generation of Android location-based apps. Automated Software Engineering **28**(1), 1–45 (2021)

[41] R.Basili, V., Caldiera, G., Rombach, H.D.: The goal question metric approach. Encyclopedia of Software Engineering, 528–532 (1994)

[42] World Health Organization Official Website: Burden of COPD. [Online]. Available: https://www.who.int/respiratory/copd/burden/en/. Accessed: 27 July, 2020

[43] Vogelmeier, C.F., Criner, G.J., Martinez, F.J., Anzueto, A., Barnes, P.J., Bourbeau, J., Celli, B.R., Chen, R., Decramer, M., Fabbri, L.M., *et al.*: Global strategy for the diagnosis, management, and prevention of chronic obstructive lung disease 2017 report. GOLD executive summary. American Journal of Respiratory and Critical Care Medicine **195**(5), 557–582 (2017)

[44] Harris, R.E.: Epidemiology of Chronic Disease: Global Perspectives. Jones & Bartlett Learning, MA (2019)

[45] Tomasic, I., Tomasic, N., Trobec, R., Krpan, M., Kelava, T.: Continuous remote monitoring of COPD patients-justification and explanation of the requirements and a survey of the available technologies. Medical and Biological Engineering and Computing **56**(4), 547–569 (2018)

[46] de Miguel-Díez, J., Hernández-Vázquez, J., López-de-Andrés, A., Álvaro-Meca, A., Hernández-Barrera, V., Jiménez-García, R.: Analysis of environmental risk factors for chronic obstructive pulmonary disease exacerbation: A case-crossover study (2004-2013). PLoS One **14**(5) (2019)

[47] CDC Official Website: Facts about hypertension. [Online]. Available: https://www.cdc.gov/bloodpressure/facts.htm. Accessed: 27 July, 2020

[48] WHO Official Website: Hypertension. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/hypertension. Accessed: 27 July, 2020

[49] WHO Official Website: Covid-19. [Online]. Available: https://www.who.int/health-topics/coronavirus#tab=tab_1. Accessed: 27 July, 2020

[50] Rechtman, E., Curtin, P., Navarro, E., Nirenberg, S., Horton, M.K.: Vital signs assessed in initial clinical encounters predict covid-19 mortality in an nyc hospital system. Scientific Reports **10**(1), 1–6 (2020)

[51] CDC Official Website: What is Diabetes? [Online]. Available: https://www.cdc.gov/diabetes/basics/diabetes.html. Accessed: 11 Aug, 2021

[52] CDC Official Website: Type 2 Diabetes. [Online]. Available: https://www.niddk.nih.gov/health-information/diabetes/overview/what-is-diabetes/type-2-diabetes. Accessed: 11 Aug, 2021

[53] Kung, D.: Object-oriented Software Engineering: an Agile Unified Methodology. McGraw-Hill Higher Education, MA (2013)

[54] ISO Official Website: Usability definition. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en. Accessed: 27 July, 2020

A Appendix

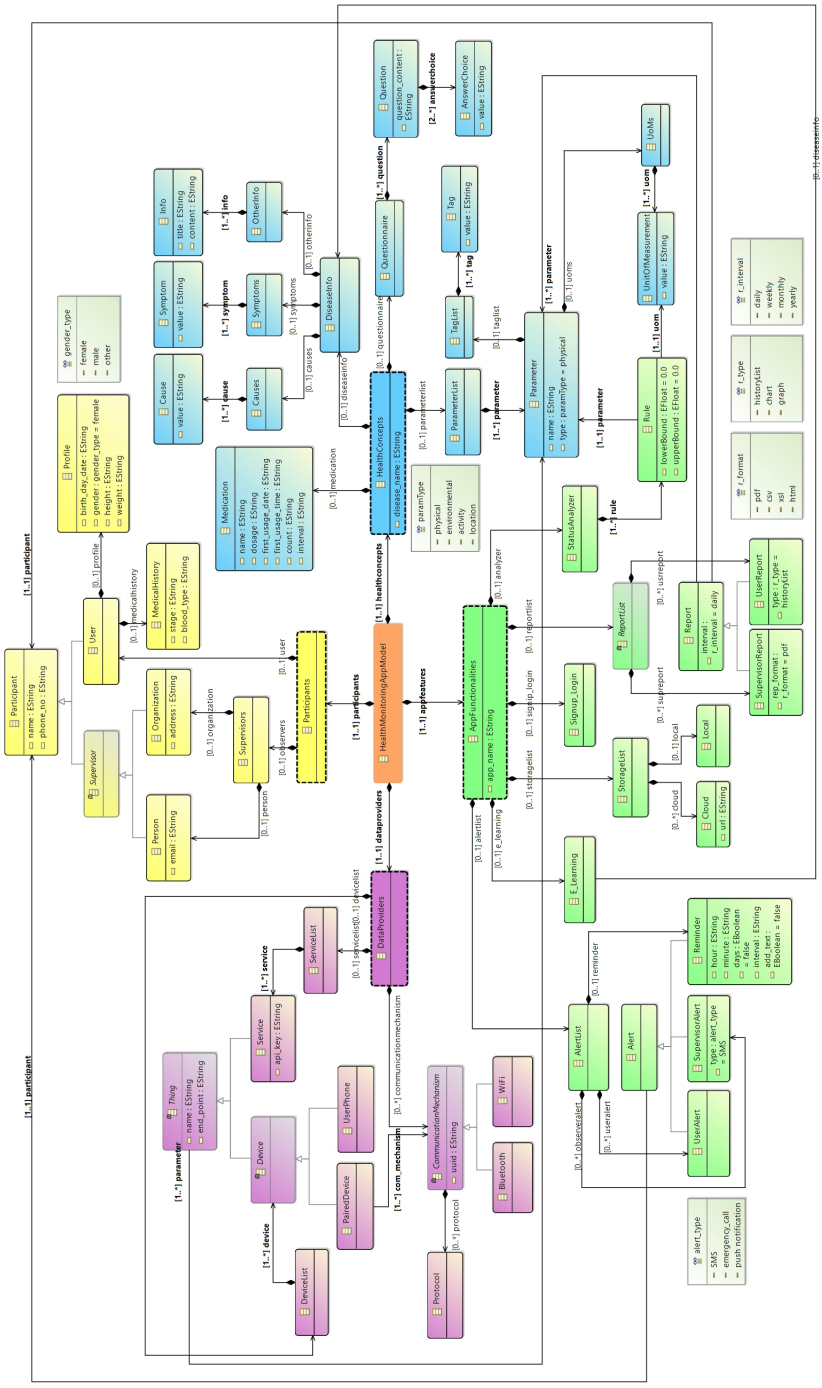

Fig. 17 Health monitoring app generation metamodel (in detail)