International Journal of Distributed Systems and Technologies
Volume 14 • Issue 1

Applying Machine Learning and Model-Driven Approach for the Identification and Diagnosis Of Covid-19

Mohammed Nadjib Tabbiche, University of Djillali Liabes, Sidi Bel-Abbes, Algeria
Mohammed Fethi Khalfi, University of Djilali Liabes, Sidi Bel Abbes, Algeria*
Reda Adjoudj, University of Djillali Liabes, Sidi Bel-Abbes, Algeria

ABSTRACT

Ubiquitous environments are not fixed in time. Entities are constantly evolving; they are dynamic. Ubiquitous applications therefore have a strong need to adapt during their execution and react to the context changes, and developing ubiquitous applications is still complex. The use of the separation of needs and model-driven engineering present the promising solutions adopted in this approach to resolve this complexity. The authors thought that the best way to improve efficiency was to make these models intelligent. That's why they decided to propose an architecture combining machine learning with the domain of modeling. In this article, a novel tool is proposed for the design of ubiquitous applications, associated with a graphical modeling editor with a drag-drop palette, which will allow to instantiate in a graphical way in order to obtain platform independent model, which will be transformed into platform specific model using Acceleo language. The validity of the proposed framework has been demonstrated via a case study of COVID-19.

KEYWORDS

Concrete Syntax, COVID-19, Graphics Editors, Internet of Things (IoT), Machine Learning, MDE, SIRIUS, Smart Healthcare System, Supervised Learning, Ubiquitous Systems

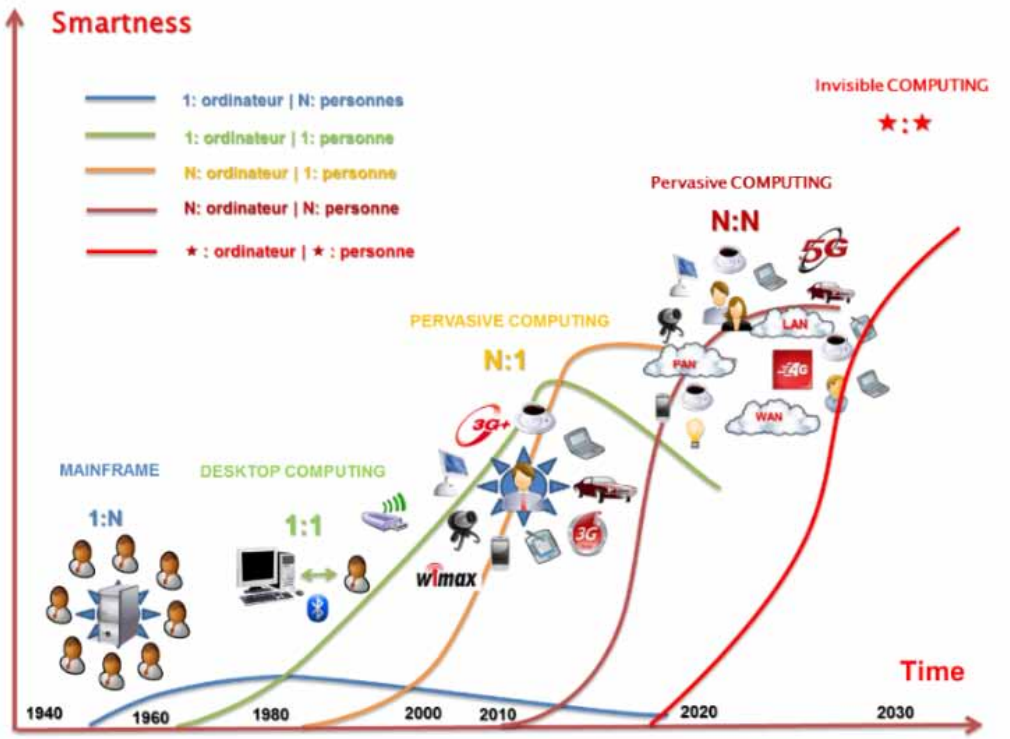
1. INTRODUCTION

Several fields are increasingly moving towards the adoption of ubiquitous technology. The term "ubiquitous computing" was proposed in 1991 by Weiser (1991). This vision describes environments filled with miniaturized computing devices embedded into everyday objects and communications infrastructures. By 2030, future networks will emerge; we will forge links with invisible life and nanoscopic nature (Khalfi & Benslimane, 2013; Lalanda, 2021), see Figure 1.

The proposed context models are very numerous and diverse. They must ensure efficiently rapid management of contextual COVID-19-Tracer and events, which is rarely the case, (Khalfi &

DOI: 10.4018/IJDST.321648 *Corresponding Author

Figure 1.
Evolution of ubiquitous computing (Khalfi & Benslimane, 2013)



Benslimane, 2015). These models evolve with the technologies offered on the market (Figure 2). Thus, a new vision is needed to minimize these technological dependencies.

This requires the use of engineering approaches and the use of high-level generic models supported by meta-models that facilitate the design of applications and allow the support of all ubiquitous features (Trajano et al., 2020).

We present two theoretical and practical contributions. The theoretical contribution concerns the design of a new intelligent meta model based on the EMF standard by combining it with machine learning.

The second contribution is a Java graphical modeling tool with a drag-drop palette have been provided to allow modeling and visualization of simple and complex ubiquitous scenarios. The proposed graphical model makes it possible to model Android applications, then generate the corresponding code. To ensure the capacity of our development process and the proper functioning of our intelligent framework, we report on the development of an Android mobile application in the field of health. The proposed model allows tracking, contextualizing, and anticipating the risks of COVID-19. To ensure the capacity of our development process and the proper functioning of our intelligent framework, we report on the development of an Android mobile application in the field of health.

The proposed model allows tracking, contextualizing, and anticipating the risks of COVID-19. Noting that our approach is generic and can be applied to any field of application. Classification of COVID-19 patients is offered into two different categories i.e., Infected, and Uninfected based on six different symptoms as input for ANN.

This paper is organized as follows: Section 2 introduces the status of our work. Section 3 details the various contributions related to the subject field resulting in a synthesis. Section 4 details the architectural model of the proposed system. Section 5 presents an instance of the COVID-19 case study MetaModel. Section 6 present discussion of results. Section 7 takes stock of our contributions and presents the research perspectives.

2. BACKGROUND: TOOLING AND MACHINE LEARNING FOR UBIQUITOUS APPLICATIONS

2.1 Technologies of Ubiquitous Systems

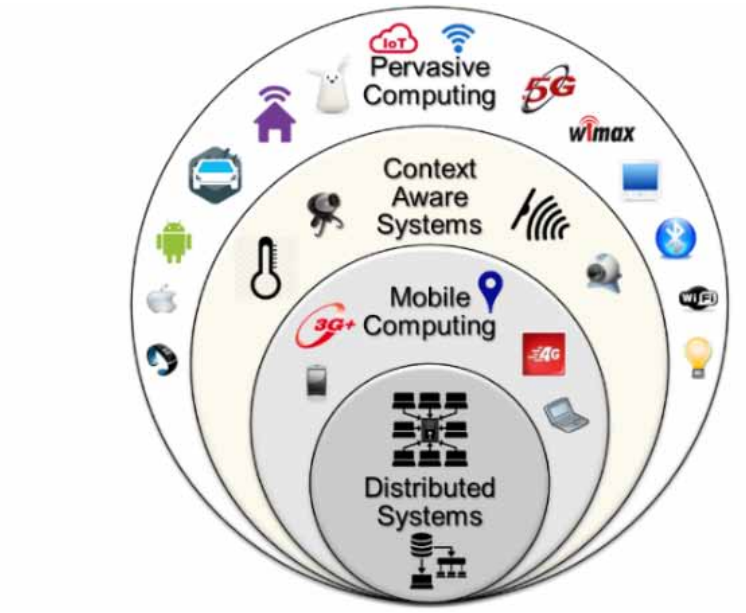
Ubiquitous computing results from the convergence of the fields of mobile computing and distributed systems (Figure 2). Distributed systems are where personal computers and local area networks meet. They enable the sharing of capacities and resources through a network and communication infrastructure. This is the first step towards pervasive computing by introducing the ubiquity of information.

Mobile computing allows the user to manage information from a mobile terminal (Tabbiche, 2021). It is the result of the integration of cellular technology and the web to give users the possibility of accessing information from anywhere and using different equipment of small size and low cost (Khalfi & Benslimane, 2013). Computers have become smaller and more powerful. They are also embedded in the objects of daily life (objects enriched with information processing capacities) (Khalfi & Benslimane, 2014). Thanks to wireless technologies, objects are interconnected and communicate for any exchange of information. In this case, we're talking about ubiquitous computing, where information can be accessed from anywhere and at any time (Khalfi & Benslimane, 2014).

2.2 Machine Learning for Ubiquitous Application

Machine learning is a branch of artificial intelligence that studies how machines can learn to recognize patterns from a COVID-19-Tracerbase or sensors (collected contextual COVID-19-Tracer)

Figure 2.
Development of ubiquitous computing (Khalfi & Benslimane, 2013)



for intelligent decision-making, (Alballa & Al-Turaiki, 2021). Machine learning techniques are constantly evolving, especially with the appearance of new technologies, in particular, "Ubiquitous Computing" and "IoT" connected objects. The dynamic nature of these constantly evolving systems has prompted researchers in the field to develop algorithms and programs that automatically detect patterns (patterns) in the COVID-19-Tracer collected, making them capable of deciding with each new situation whether to adapt to ubiquitous environments.

The contextual COVID-19-Tracer collected can be of different formats: images, signals, and texts; and can come from different sources: hybrid sensors (thermal, photovoltaic, etc.). ii) external and environmental sensors (such as humidity, temperature, light, cameras, and so on), and iii) portable sensors (such as smartphones and Smart Watches).

There are different types of algorithms to use, each depending on the task to be accomplished and the type of COVID-19-Tracer available (a lot or little COVID-19-Tracer) and the problem you are trying to solve (recognition, classification, etc.). These algorithms are divided into four classes of learning techniques, namely supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning (Khalfi & Benslimane, 2014).

2.2.1 Supervised Learning for Ubiquitous Application

This type of learning allows you to learn from a set of previously labeled examples by an expert Mohammadi et al., 2020; Fedushko & Ustyianovych, 2020). Once the learning of the model is completed, the learner will be faced with a set of tests that he will have to predict (Figure 3). This learning family is used to perform prediction, diagnosis, or classification. Various algorithms, including the Naive Bayes algorithm, support vector machines (SVM), and neuronal networks, can be used to implement supervised learning (Elhannani et al., 2022).

In the application context that interests us in this work, this type of learning can be used as a reasoning entity for our medical application. It can help contextualize, anticipate, and predict the risks of COVID-19.

2.3 MDE (Model-Driven Engineering) for Ubiquitous Application

MDE is based on the use of models throughout the software wrapping process (Rademacher et al., 2020). Any model is defined using a rigorously defined modeling language to meet the specific needs of a domain. This process is seen as a set of model transformations, where each transformation takes models as input and produces models as output until obtaining executable artifacts (Khalfi & Benslimane, 2014). The MDE implements the IDM concepts (Figure 4).

Figure 3.
Supervised learning approach

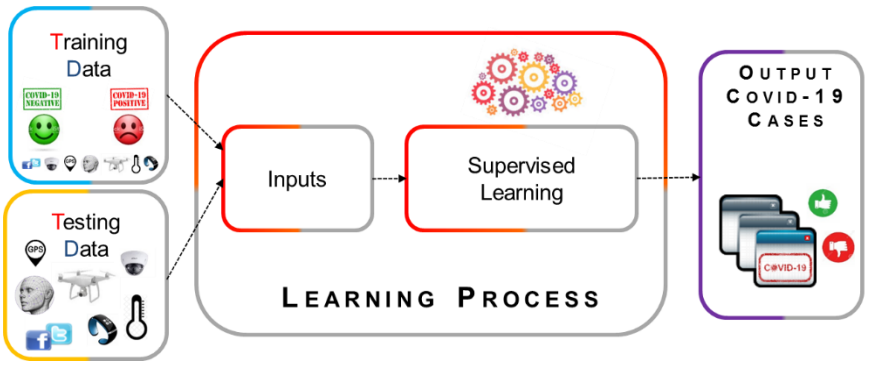
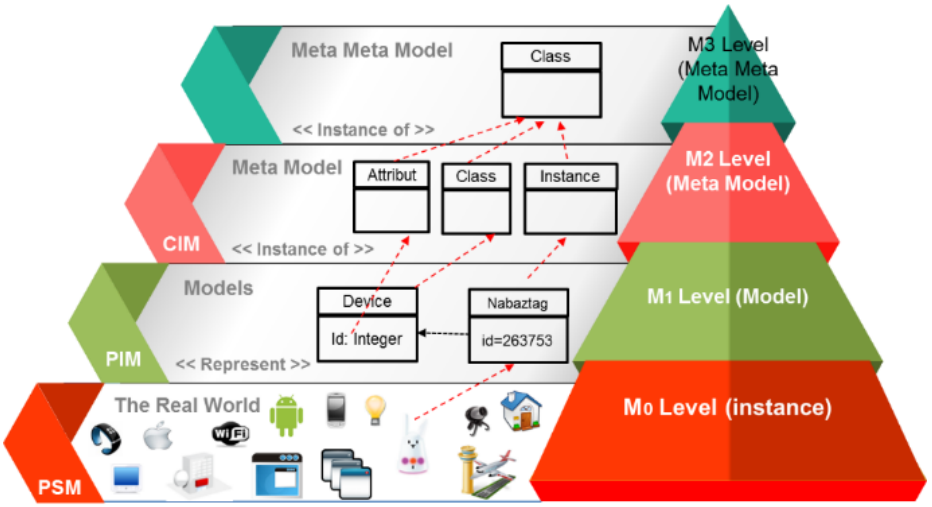


Figure 4.
MDE model levels (Mettouris & Papadopoulos, 2019)



3. RELATED WORKS

We have selected works that address the contributions of IDM to the design of ubiquitous applications. We employ a set of criteria. Some are inspired by the works that have been reviewed, while others have been identified based on the requirements of our problem (Table 1; Moradiet al., 2020). Develop a framework called CaaSSET, which stands for Context as a Service Set of Engineering Tools (Baddour et al., 2019). Propose an integrated and formal context modeling approach for intelligent systems operating in context-sensitive environments (Dorndorfer et al., 2019). Present a domain-specific modeling language for context and a model-driven architecture-based approach for mobile context-aware apps (Mettouris & Papadopoulos, 2019). Apply the Model-Driven Development paradigm to the physical commerce recommendation domain by defining a UbiCARS DSML (Li et al., 2018). Propose a DSL for model-driven development of AOCA applications. They analyzed the conceptual model of such applications and designed the AocML abstract syntax and concrete syntax (Jaouadi et al., 2018). Present a model-based approach that addresses the development of context-aware applications that supports all development phases of context-aware systems (Boudaa et al., 2017). Propose a generic model-driven approach for context-aware service-based application engineering with a software development methodology (Duarte, 2014). The adoption of LoCCAM middleware platforms for context-aware systems is a well-known solution used to overcome such problems (Fedushko & Ustyianovych, 2020) describes an algorithm of actions on machine learning using, determining the student's success level and analyzing the obtained data (Mettouris & Papadopoulos, 2019). In this paper, UbiRS4Tourism Model Driven Framework for tourism recommendation domain is proposed. The Framework utilizes a model driven methodology and defines a novel graphical Domain Specific Modelling Language, aiming to reduce the complexity of the development for no expertise.

To characterize and compare the work related to our proposal, we use a set of criteria. Some are inspired by the reviewed works, and others were identified based on the requirements of our problem (Tabbiche, 2021). Table1:

- Level of Abstraction:
  - Some approaches have adopted a graphical concrete syntax from other textual.
- Conformance MDE:

Table 1.
Syntheses

Approach		S. Fedushko, 2020	H Moradi 2020,	A. M. Baddour et al 2019	Dorndorfer J. 2019	C. Mettouris, 2019	Li, X., Tao 2018	ContextServ, 2018,	Jaouadi 2019	B.Boudaa 2016	Duarte, P.A., 2014	Our Proposition
Application Domain		Educational institutions	Tourism context service	Smart Meeting Rooms	printer management	physical commerce and tourism recommendation	Smart meeting room	Smart Shopping Guide	Healthcare	Healthcare SAMU	Android projects	Covid-19
Tool for development		R programming language	EMF + GMF + Acceleo	EMF + GMF + Papyrus	Obeo Designer	Serius + WooCommerce + Drupal	PAOC + Xtext	Extended ArgoUML	EMF Eclipse Ecore editor	EMF Eclipse	EMF Eclipse Ecore editor	EMF + Sirius+ MachineLearning
syntax Abstract		×	EMF (Ecore)	EMF (Ecore) + UMLClass Diagram	EMF (Ecore)	EMF (Ecore)	UMLClass Diagram	ContextUML	EMF (Ecore)	UMLClass Diagram + ODM	EMF (Ecore)	EMF (Ecore)
Context management	**Context language modeling**	Mathematical Modeling	EMF (Ecore)	EMF (Ecore)	DSML SenSoMod	DSML for UbiCARS, UbiRS4Tourism	Specific textual DSL	UML (Extend ContextUML	XML	Ontology& UML	DSML LoCCAM	EMF (Ecore)
	Context Reasoning Smartness	Machine Learning	ECA (Event of condition do action)	First-order logic (FOL)	ECA (Event of condition do action)	CARSKIT Engine	Jena Rule-Based Reasoner	ECA (Event of condition do action)	ECA (Event of condition do action)	SWRL (Semantic Web Rule Language)	ECA (Event of condition do action)	Supervised Machine Learning
	Context modeling	Mathematical	Graphical	UML Class Diagrams + OCL	Graphical	Graphical	UML & Ontology	UML	XML	Ontology	UML	Graphical
	Context history	Online sociological Survey	context repository	Covid-19-Tracerbase	×	Database Resource	Covid-19-Tracerbase	XML Repository	Context Repository	×	XML Repository	XML, XMI repository
Adaptation	**services**	EDA (Exploratory data analysis)	Context Triggering	Context Triggering	Context Triggering	adaptation Algorithms	Context Triggering	Context Triggering and binding Behavior adaptation	Context Triggering	Context Binding, Triggering and Behavior Adaptation	Context Triggering	Supervised Machine Learning
	Content (Covid-19-Tracer)	Data collection Process (online sociological survey)	push/pull	×	×	×	push/pull	Context Monitoring Process	Context control	Observer module	Publish/ Subscribe	Push/Event
Service modeling		Process tasks	Context web service	Focus Specification	×	Service Recommandation System (RS)	AOCA Services	WSDL, BPEL specification	Task Services	Code FraSCAti	OSGi Services	Event-ML-Action

  - Conformance with the MDE standard is partially or completely ignored.
- Service Modeling:
  - The work offers a variety of service modeling (Context web service, WSDL, BPEL Specification, Code FraSCAti...etc.).
- Decoupling Context/Business Logic:
  - The majority of approaches have realized the partial decoupling between the context management and the business part.
- Transformation: few proposed approaches have implemented transformation techniques.
- Context management:
  - The proposed context metamodels do not cover the main contextual entities.
  - Most of the approaches used the ECA (Event of Condition Do Action) reasoning model.
  - There have been few works that address the issue of contextual quality.
- Adaptation:
  - No approach has dealt with adaptations on the User Interface (HMI) side.
  - Most of the approaches have adopted a simple adaptation technique called "ECA Rules", which does not satisfy one of the requirements and the nature of ubiquitous systems, which is uncertainty and frequent change of context.
- Privacy and Security: Although security and privacy (confidentiality) are essential aspects for ubiquitous applications, they are only considered by very few approaches.

4. METHOD

Our work focuses on the design of ubiquitous applications from an IDM approach. These applications evolve in information-rich environments; therefore, they must adapt their behavior according to the variations of the contextual information to offer services adapted to the user.

It is assumed that the elements of the context must be treated independently of the other constraints because they present specific and highly variable characteristics (mobility, heterogeneity, etc.) (Tabbiche, 2021).

The separation of needs according to their nature (functional, ubiquitous, or technical) is the solution adopted in our approach. This separation makes it possible to define a business model independent of any technology platform and to automatically generate code for the chosen platform. Thus, the development process will not be called into question because of the possible evolution of technological support.

We will explain step-by-step the whole process that we have developed to arrive at our solution, (Tabbiche, 2021):

1. The reference architectural model.
2. The context metamodel consists of creating the PIM model independent of all execution platforms, based on an abstract grammar with the EMF tool. The proposed abstract syntax is supplemented by a concrete syntax: the development of a graphical modeling workshop using the Eclipse Sirius tool for the graphical representation of PIM models.
3. Defining OCL constraints to capture model constraints that could not be captured by metamodels.
4. Using Acceleo, write code and define a model to text transformation (M2T). The generated code is used to design Android applications. The Android project can be generated in the same Eclipse instance as the modeling environment. Thus, the designer/developer does not need to change the environment to be able to test the generated application.

4.1 Architecture

In addition to the technical separation offered by the MDE standard, our approach is based on the principle of explicitly separating the business part (functional) and the ubiquitous part (contextual) and, from the conceptual modeling phase (PIM and UbiPim) (Tabbiche, 2021). These choices motivated the creation of a level of ubiquitous development independent of technical and functional needs (see Figure 5). Any type of concern (business, technical, or contextual) will be expressed using a very specific model (respectively: PIM, UbiPim, PM, and PSM) (Khalfi & Adjoudj, 2021).

We wanted our framework to be based on standards in this field and to have a chain of tools allowing us to describe our meta-model and be capable of automatically generating code from it. This chain mainly includes tools running in an Eclipse IDE (Integrated Development Environment): EMF, Sirius, Xtext, and Acceleo (Ozkaya, 2021). When we combine them, we get Figure 6, which presents the technical architecture of our workshop and how the different tools are interconnected.

4.2 A Context Meta-Model

4.2.1 Abstract Syntax Specification

As part of this research work, we have opted for EMF to define the metamodel or the abstract syntax of the language (expressed in Ecore), (Khalfi et al., 2021). Figure 8 illustrates our metamodel (Tabbiche et al., 2021). Ecore is a high-level graphical language that specifies the abstract syntax for creating metamodels satisfying the EMOF standard. It uses the graphical syntax of UML, which is relatively well known and therefore quite intuitive. It was thought to make these models intelligent by combining machine learning with EMF (Moin & Rössler, 2018; Moin, 2021; Hartmann, 2019). The ubiquitous system is represented by Mohammed Nadjib Tabbiche (2021):

- **The element (System):** It can contain other systems thanks to the "OwnedSystem" relationship. The "ItsEntity" aggregation relationship specifies that an instance of a system is composed of several contextual entities involved in an ubiquitous application.

Figure 5.
New pyramid MDE for ubiquitous application

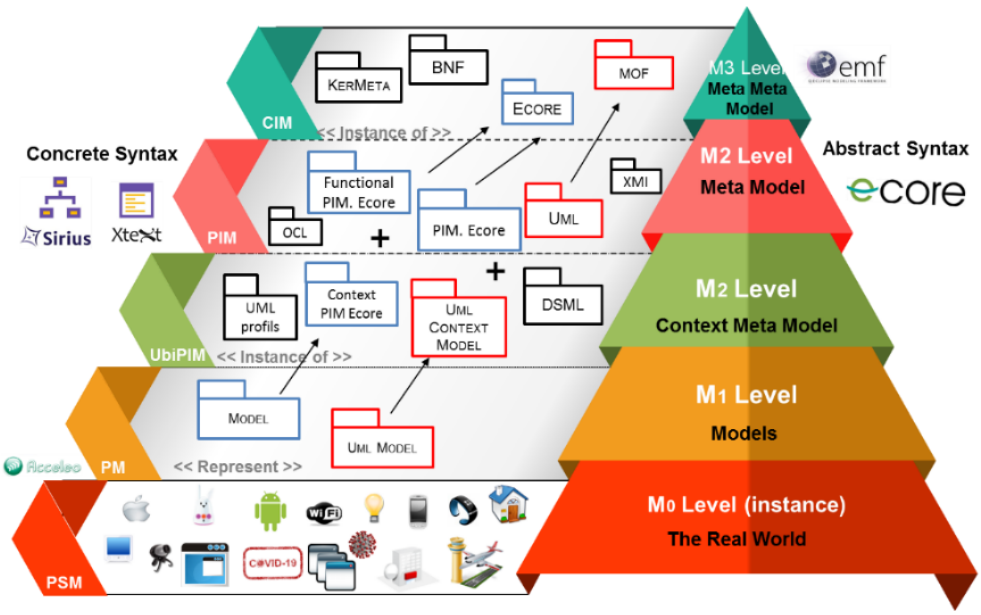
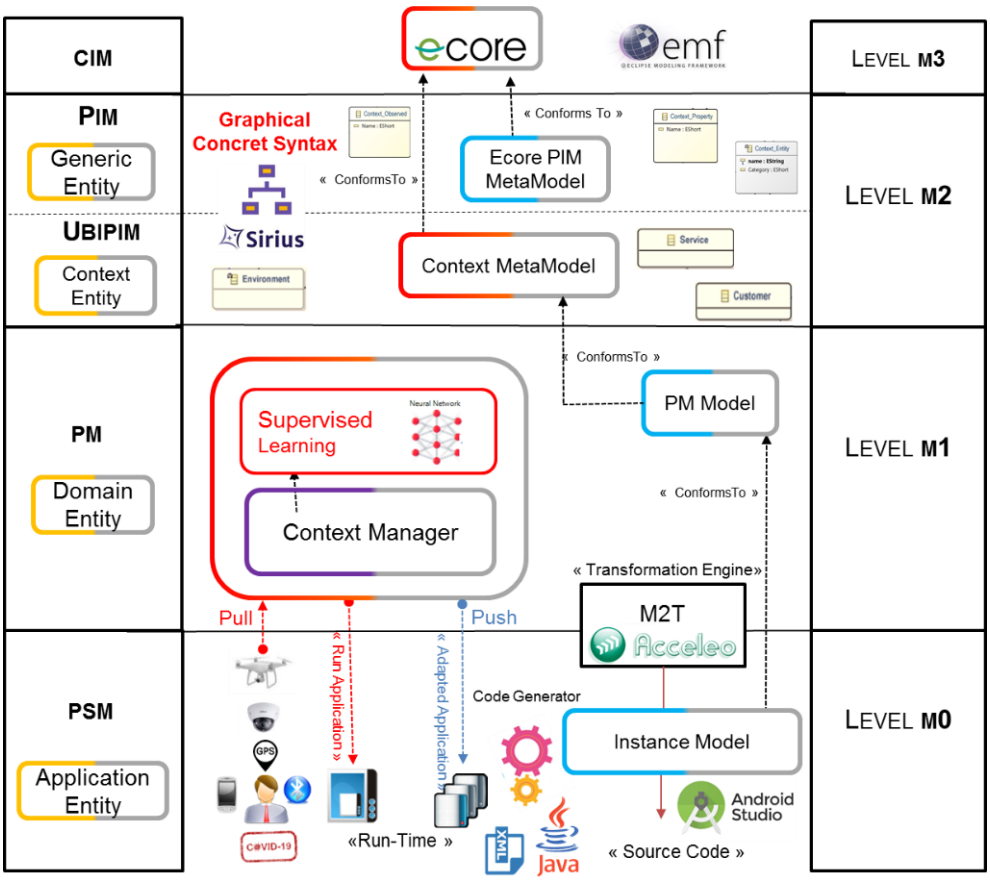


Figure 6.
Architecture of our approach



- **ContexteEntity:** It is the central class of this metamodel. It represents entities identified by the relationship "ItsEntity" to the system. A contextual entity is a concrete representation of a real-world object deemed relevant to describing a domain. It can be composed of other entities through the "OwnedEntity" relationship.

The operation of ubiquitous applications requires the collection of all relevant context information from various sources. There are two categories of context:

- **Atomic Context:** It represents low-level contextual information that is not linked to another context, and it can be obtained by: Physical sensors (GPS, Wi-Fi, Bluetooth, NFC), Virtual sensors (web services), Social sensors Flickr service. Microblogging (Twitter).
- **Composite Context:** It represents high-level contextual information; it consists of multiple contexts. It can be obtained by: Logical Sensor.
- **Context Source:** It represents the resources from which contextual information is retrieved. It makes it possible to link the contextual elements to their sources to identify the method used for the acquisition of the contextual data. The acquisition of this data can be: static, profiled, sensing, derived from other data (derived), or learned.

- **ContextObservable:** They describe the data collected by the sensors. It contains "Value" as a value of a simple type (Integer, Long, Float, String, or Boolean) measured by the sensor. Observable contextual information can be either atomic or composite. The acquisition of the contextual information can have a temporal constraint, which can be either an absolute interval or an expiry date expressed in this metamodel by the "Constraint" metaclass.
- **ObservationMode:** A sensor's role is to take measurements and then send the data to its customer(s) or allow its customer(s) to come and retrieve it. The obtaining of the data can be done according to the observation mode ("ObservationMode") By notification ("Notify" or "Push"), it is the sensor that sends the data each time this information changes to notify the interested entities.
- **NotificationEvent:** The "Notify" mode is divided into three modes: "SimpleNotify", "PeriodicNotify", and "ConditionalNotify".
- **ContextProperty:** It describes the properties of the ContextEntity class. Each ContextProperty is described by its name and type.
- **ContextRepository:** Applications can use not only the actual context but also the perceived context to adapt their behavior to better interact with users. Thus, we store contexts continuously, as they occur, in a database.
- **ContextAssociation:** It is used to describe the relationship between two entities or the relationship between the context entity and its properties. Each association is described by a name.
- **CAMechanism:** To consider the uncertainty of ubiquitous environments, this part allows the modeling of adaptation actions according to context, and all these actions are represented by the CAMechanism class.

We decided to introduce techniques based on supervised learning:

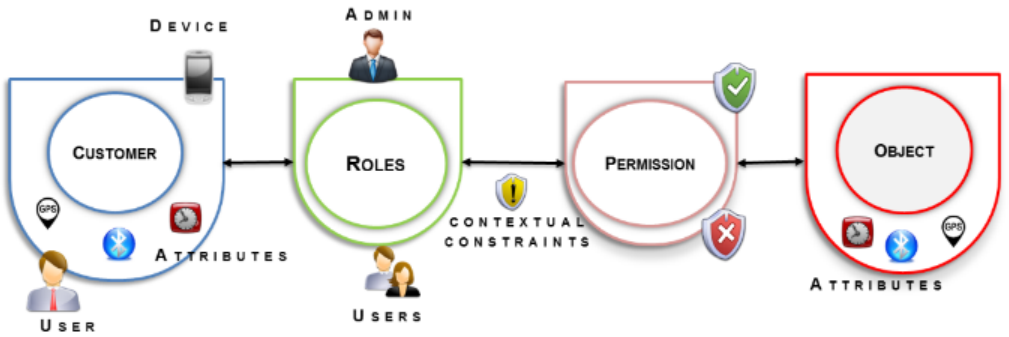
- **ContextMachineLearning:** We propose a meta-model that collects the machine learning process's common entities. It includes a set of techniques based on supervised learning and actions to be predicted and respectively represented by the classes "Supervised_ML_Algorithm" and "Action_Prediction" in the meta-model.
- **MachineLearning:** This class uses supervised and unsupervised learning algorithms to train models. Then, the results of each algorithm are evaluated to determine the accuracy. In the end, it selects the appropriate prediction model.
- **Inputs:** Are a collection of data that machine learning algorithms use to train the model.
- **Outputs:** Are a collection of data predicted by the trained model.
- **Scores:** To check the accuracy of learning and inference, evaluation measures (scores) are needed. The accuracy of machine learning models must be evaluated to select the best performing model for prediction.
- **Prediction Action:** The result of the prediction is returned for decision-making.

Our meta-model can also be extended by integrating the confidentiality and privacy aspects of the context source entity. Indeed, the collection of a person's contextual data (e.g., his location) must be done with the agreement of the person in question and therefore according to his preferences.

- **Contexte Security:** The extended meta-model consists of modeling access control and privacy policies embedded in a model-driven software development process. The meta-model is based on the ABAC and RBAC models without considering roles as user attributes (see Figure 7). Our model considers the contextual attributes of the Entity Costumer and the system resources that may be accessed.

The various extensions to the model relate to the incorporation of contextual constraints. Dynamic authorization requires access to be re-evaluated according to changes in the contextual information used

Figure 7.
Both RBAC and ABAC models combined



to gain access. This would require the security system to be alerted to changes in context throughout their use of the system. In this case, the system would respond to context changes by evaluating the authorization rules to determine whether the subject is still authorized.

- **QoC:** Contextual information must have very specific quality properties to be considered efficiently. It can be evaluated according to different criteria: confidence, accuracy, freshness, and resolution.

According to our model, each QoC criterion can take a value between [0] and [1]. The calculation of the QoC metrics depends on the type of contextual source (physical, virtual, and social sensors).

- **Modality:** These relationships make it possible to find out which devices are currently interacting with the user and how they interact with the user, (Quan Z. Sheng and Boualem Benatallah, 2005).
- **Mobile App:** The proposed meta-model covers the basic aspects of mobile application programming for the Android platform. The workshop will allow us to model basic concepts such as activities, elements, and the usual graphical components (Text, Buttons, List, and so on).

The View part represents the user interface of the application, the basic graphical component of the Android platform. The language provides two types of view: content (widgets) and container (layout). The different graphical components (views) can be buttons and text fields, for which we can define common attributes. The widget elements, such as labels, fields, and buttons, are grouped and arranged inside the layout elements. Screen: Represents an application screen and a displayed user interface.

One or more activities comprise an Android application. An activity is divided into two parts: an XML file and a Java file. Therefore, the metamodel of an Android application has two categories of elements: Java elements (JavaElements) and XML elements (XmlElements).

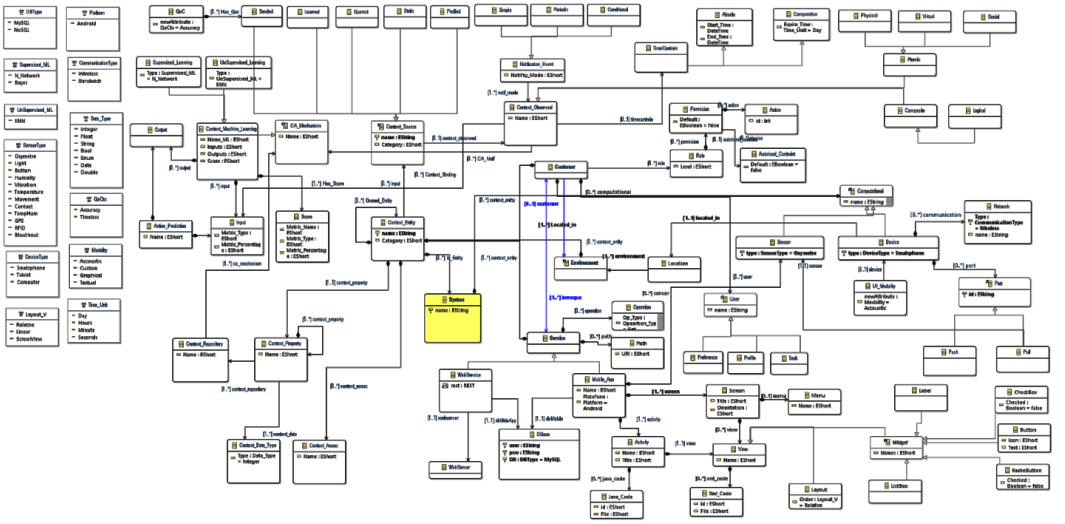
4.3 The Concrete Syntax (Graphic Modeller)

4.3.1 Design of a Graphic Concrete Syntax

The concrete syntax (CS) of a language allows the user, according to his needs, to formally express a model and then instantiate and manipulate the concepts of the abstract syntax. Each concept in the abstract syntax is associated with a graphical icon that allows the associated concept to be represented at the model editor level.

Our contribution is based on the specification of the abstract syntax formalized in the metamodel presented in the previous section to define a concrete graphical syntax via the Sirius tooling, which can be adapted to specific ubiquitous and business needs (Bedini et al., 2021).

Figure 8.
The meta model expressed in EMF (Ecore)



4.3.2 Graphics Workshop for the Representation of PIM

Sirius is an open-source framework tool that enables the design of a graphical modeling workshop based on Eclipse Modeling technologies. The modeling workshop allows users to create, edit, and visualize EMF models graphically.

In our work, we give users the tools to create representations of their future ubiquitous applications. After the design of our meta-model, we will now define a graphical editor that allows us to instantiate our meta-model in a graphical way to obtain PIM models, which will be later transformed into PSM.

4.3.2.1 The Components of the Proposed Modeler

A palette of modeling tools is created to give more flexibility to the user wishing to match the metamodel concepts presented and the editor. It groups generic entities, contextual entities, domain entities, and the associations between them.

The section presented in Figure 9 is used to group the different components to be used to generate an application. This separation provides a clear view of the components as well as the flow of connections between the components.

The main interface of the modeler is shown in Figure 9. It contains the following containers (see Figure 11):

- **ContextProvider:** The modeler allows developers to create entities and assign them to concrete virtual objects. It also offers wired and non-wired connection choices. Entities can be enriched with non-functional information (security, QoC, and user interface modality).
- **ContextAssociations:** Developers can define the relationship between the connections between the sensors and the chosen learning model (supervised: neural network or ANN): (PULL---Sensors). Developers can also define the relationship between the connections between the learning model and the end terminals (telephone or PC).
- **ContextMachineLearning:** The "Services" class offered by the Sirius project is used to make use of the learning models from the modeling phase. This class is used to handle dynamic processes that visualize changes in real-time. Figure 10 shows the use of Java Services.

Figure 9.
Main interface of the modeler

- **MobileApp:** The proposed metamodel can be used to model web and mobile (Android) applications. Thus, the graphical editor can create Android application models.

4.4 Specification of a Semantic

Abstract and concrete syntaxes are sufficient to manipulate a language graphically or textually. However, the automatic manipulation of models by tools requires a formalization of the semantics.

We used the Ecore meta-modeling language to define our meta-model, but it does not allow us to express all the constraints that models must respect and does not allow us to avoid certain ambiguities.

Figure 10.
The use of Java services: NN and Knn

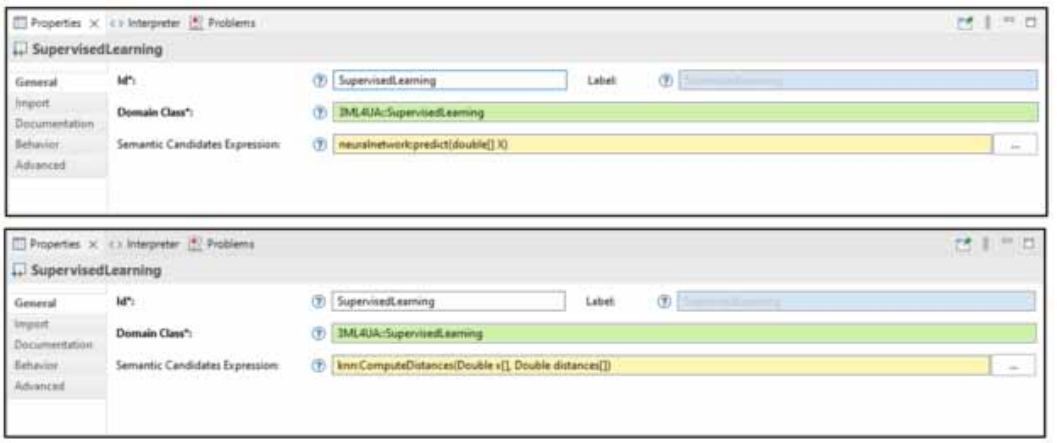
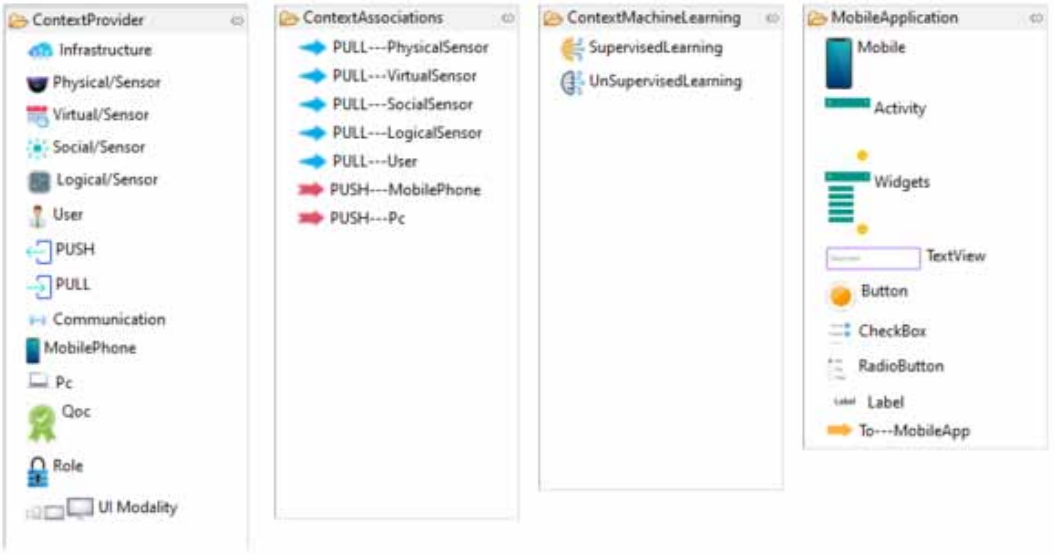


Figure 11.
Components context provider



For this, we complete the structural description of the meta-model with formal annotations and expressed constraints. OCL was used to define validation rules to check the compatibility of the created models with the metamodel.

Its implementation in the EMF environment is possible thanks to the OCLinEcore tool, which allows annotating an Ecore model. Among the OCL rules defined to ensure validation, we will mention the following (see Figure 12):

1. The data transmitted by the sensors to the ML and the data retrieved after the ML must be of the same type.
2. No two sensors should be named the same.
3. No two services should be named the same.

4.5. Source Code Generation

We are using Acceleo to automate generation from high-level modeling. Founded on the MDE approach, the general principle of Acceleo consists of applying a generic template to a starting model to obtain an output code. An Acceleo template is a ".mtl" extension file designed to configure the code to be generated on a target model.

4.5.1 Transformation Rules

To implement the code generator, the Model-to-Text transformation (M2T) is needed to generate the source code for an application that can be run on an Android device. We decided to develop first for Android as it is the most popular mobile operating system.

The folder structure generated for Android follows the project design pattern in Android Studio in the form of a tree structure. It contains several folders and some configuration files at the root. An Android project has two categories of elements: Java elements and XML elements.

Figure 12.
Some OCL validations

```
context Pull
inv: self.dataType=Push.dataType
context Push
inv : self.dataType=Pull.dataType
```

```
context Sensors
inv : Sensors.allInstances()->forAll(s1, s2|s1<>s2 implies s1.name<>s2.name)
```

```
context Action
inv : action.allInstances()->forAll(a1, a2|a1<>a2 implies a1.name<>a2.name)
```

1. **The Java elements:** See below:
   a. **Src/:** Contains the project's Java source code.
   b. **Gen/:** Contains the source code produced by the Android build tools.
2. **XML elements:** See below:
   a. **MainActivity.java:** It contains all the activities of the application. Its layout in XML is the main screen of the application.
   b. **AndroidManifest.xml:** The manifest stores' properties and parameters of the application configuration, such as the name of the application's functions, the code version, access rights, etc.
   c. **Res folder:** It contains all the resources we use as well as the XML files for the interface:
   d. **Widgets:** It contains the XML files that present the graphical interface. This file contains the declaration of the graphic components.
   e. **Values:** It contains the XML files that define constant values (strings, dimensions, colors, styles, etc.).
   f. **Drawable:** It contains any element that can be drawn on the screen: images, shapes, animations, etc. xml: The file containing the constant declarations.
   g. **Gradle:** It is a script that compiles the application and installs it on the terminal. It is responsible for importing the necessary libraries for the application.

Figure 13 illustrates the main model of transformation rules, which are performed based on templates that reside in the .mtl files.

4.5.2 Design of the Code Generator

After creating the Acceleo project and choosing the source meta-model (named 3ML4UA (for Meta Modele Machine Learning for Ubiquitous Application), we will then create a main module of the code generator "EntityGenerator.mtl" for the Android application. It will call the other modules (MobileApp, CompUser, and Communication) and templates, allowing us to generate code in an Android Studio project imported in Eclipse and which must be executed on an Android Device. Figure 14 illustrates the main module of the Code Generator.

The other modules and templates called by the main module "EntityGeneraor" are as follows (see Figure 15):

4.5.2.1 Module MobileApp.mlt

Figure 13.
The different stages of generating an Android application

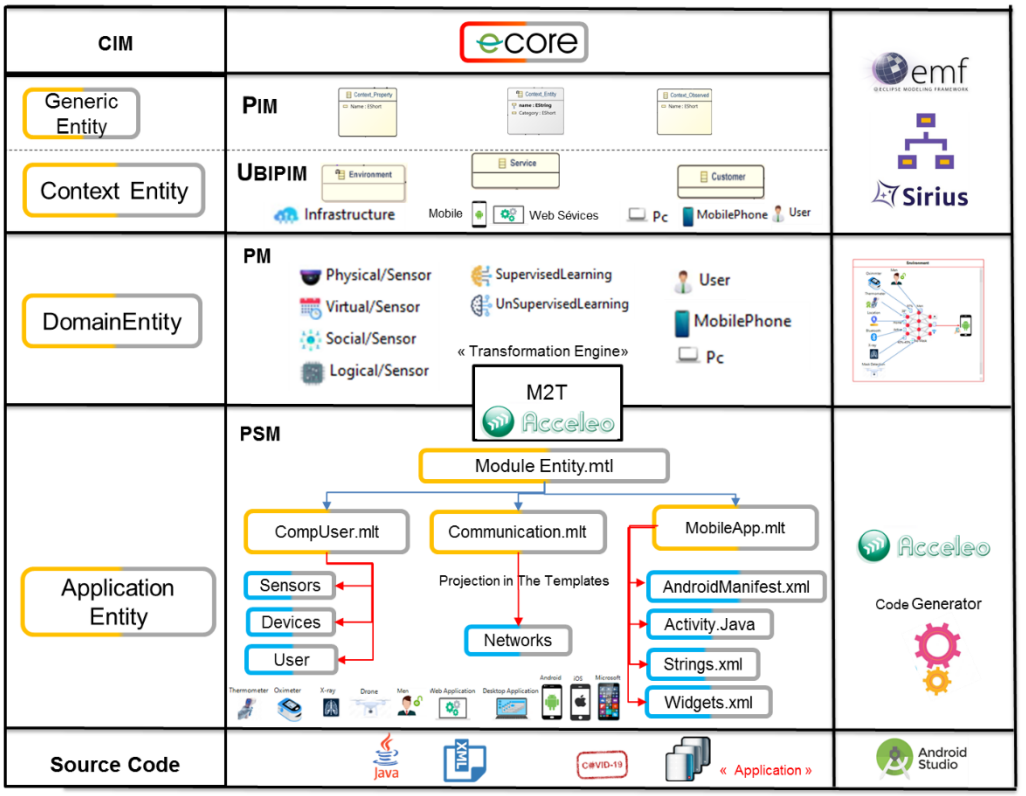


Figure 14.
The main module of the EntityGenerator.mtl code generator

Figure 15.
Template of generated elements

- **Template generateActivity:** It is used to generate the "src" directory. The Java classes are all generated in the "src" directory of the Android project. The user can then directly execute his application.
- **Template generateView:** It generates the .xml file that contains the widgets. A widget is composed of different graphical components (views). We will use the iteration structure to call all the components.
- **Template generateAndroidManifest:** It is used to generate the AndroidManifest.xml file. It is located at the root of any Android project and it is a sort of complete description of the application (components, activities, services, permissions, content providers, etc.). It is a file that every Android application must contain (see Figure 16).
- **Template generateResources:** It is responsible for creating the values directory. The resource values that are defined in the PIM part will be transformed into an .xml file such as 'res/values/strings.xml'.

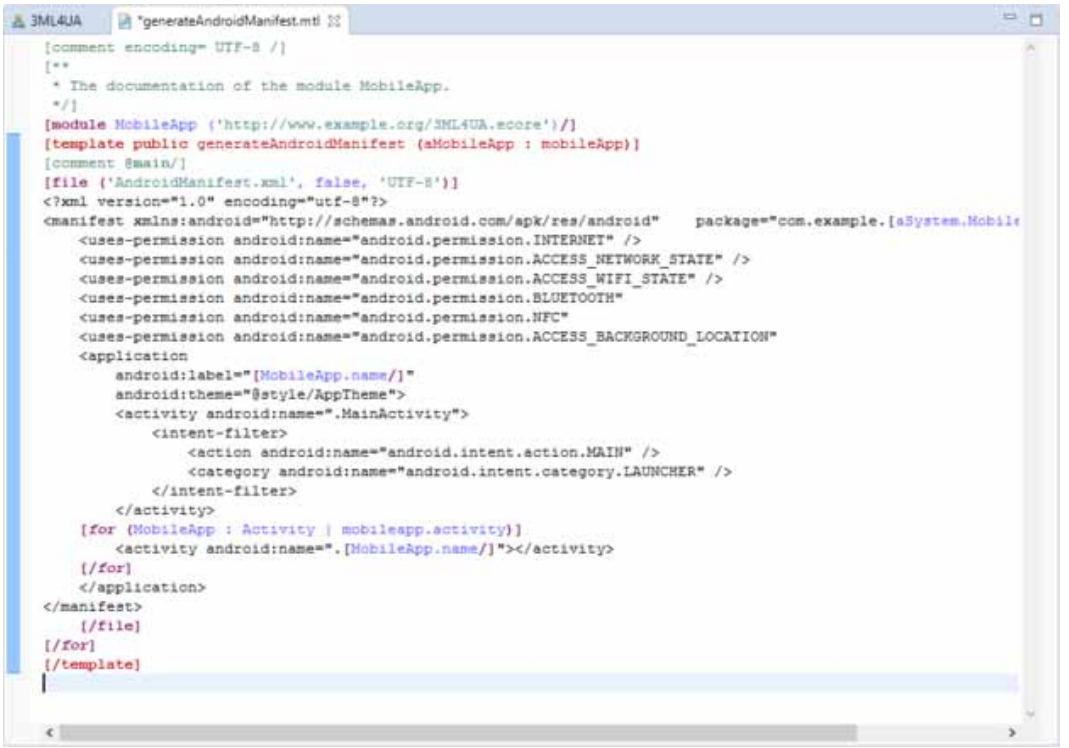
4.5.2.2 Module CompUser.mlt

- **Template GenerateSensors:** It is used to generate the various sensors: physical sensors (GPS, Wi-Fi, Bluetooth, NFC), virtual sensors (web services), Social sensors (Twitter), etc. (see Figure 17).
- **Template GenerateUsers:** It is used to generate various devices such as smartphones, PCs, tablets, etc.
- **Template GenerateDevices:** It is used to generate the various devices such as smartphones, PCs, tablets, etc.

4.5.2.3 Module Communication.mlt

- **Template GenerateNetworks:** It is used to define the different networks used.

Figure 16.
Template generateAndroidManifest



5. RESULTS AND DISCUSSION

5.1 Case Study Presentation: COVID-19

Our job is to generate the simulated model based on the concepts of model-driven engineering. In order to illustrate our approach, we will take an example of a topical application that could limit the spread of the virus by identifying chains of transmission. The idea would be to warn people who present symptoms of COVID-19 or who have been in contact with a patient who tested positive to be able to be tested themselves and, if necessary, to be taken care of very early or else to confine themselves.

5.2 Proposed System Architecture

The use of data and IoT to monitor the population is not new, but the phenomenon has accelerated considerably with the COVID-19 health crisis.

The architecture is primarily based on continuous, real-time observation of patients with probable COVID-19 (see Figure 18). The current study uses a sensor network to collect user-personal information and symptoms of infection. Breathing difficulty is the most common symptom among all symptoms during a COVID-19 infection.

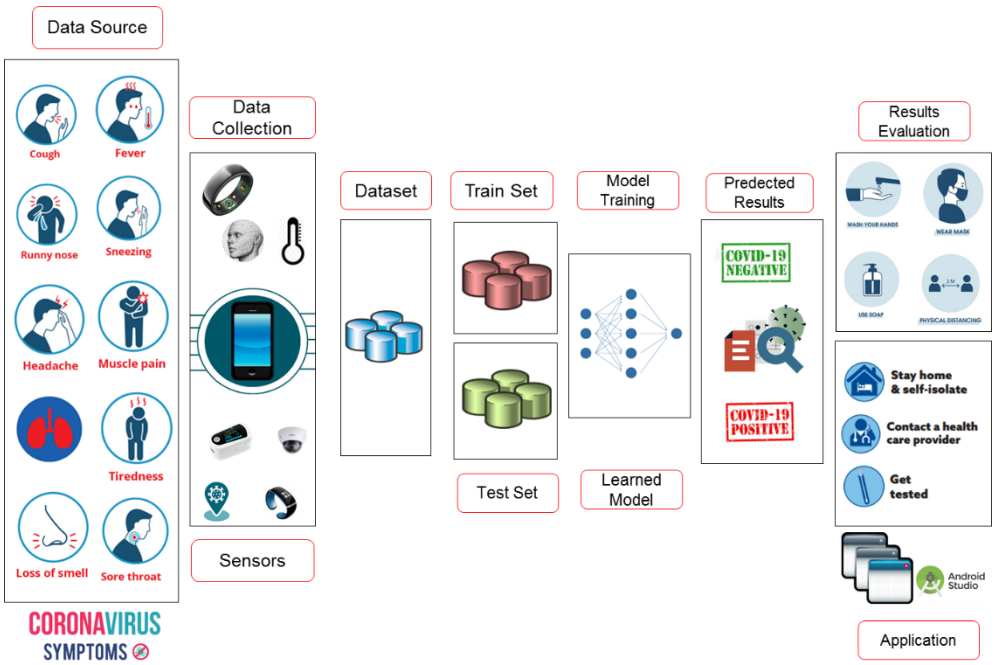
The proposed architecture uses machine learning techniques that allow diagnosis by performing an analysis based on the symptoms collected by the various sensors. The recorded data is separated into training and test sets. The trained model is tested using other expert-validated datasets.

The results of the predictive model are communicated to the medical experts to obtain a valid prescription. Based on the results, the expert suggests whether the patient should be tested or hospitalized, take medication, or be isolated. The results are useful for people whose predicted results

Figure 17.
Template generateSensors

```
3ML4UA   *GenerateSensors.mtl
[comment encoding= UTF-8 /]
[**
 * The documentation of the module MobileApp.
 */]
[module Computational ('http://www.example.org/3ML4UA.ecore')/]
[template public GenerateSensors (aSensors : sensors)]
[comment [@main/]
[file (Computational.concat('Sensors.java'), false, 'UTF-8')]
package com.example.[ Sensors.name.toLower()/];
[/file]
[/template]
 [for (sensor : Sensor | s.CompUser)]
        [for (Psensor : PhysicalSensor | ps.Sensor)]
                [if (Psensor.type.toString()='Oximeter')]
                                        int data[Psensor.name/] = Read([Psensor.name/]);
                                                [/if]
                [if (Psensor.type.toString()='Camera')]
                                        int data[Psensor.name/] = Read([Psensor.name/]);
                                                [/if]
                [if (Psensor.type.toString()='Bluetooth')]
                                        int data[Psensor.name/] = Read([Psensor.name/]);
                                                [/if]
                [if (Psensor.type.toString()='Temperature')]
                                        int data[Psensor.name/] = Read([Psensor.name/]);
                                                [/if]
                [if (Psensor.type.toString()=Thermometer)]
                                        int data[Psensor.name/] = Read([Psensor.name/]);
                                                [/if]
              [for (Vsensor : VirtualSensor | vs.Sensor)]
                                                [if (Vsensor.type.toString()='Calendar')]
                                        int data[Vsensor.name/] = Read([Vsensor.name/]);
                                                [/if]
                                                [if (Vsensor.type.toString()='WebService')]
                                        int data[Vsensor.name/] = Read([Vsensor.name/]);
                                                [/if]
```

Figure 18.
Architecture of the proposed system



are negative. The expert can suggest that the patient respect the barrier gestures, wear a mask, and put on the hydro-alcoholic gel.

5.3 Modeling our COVID-19 Case Study With the Graphic Editor

For the TraceCOVID-19 case study modeling, we used the Sirius tool. The first version of our modeler has been implemented. It is based on the Eclipse Modeling Framework tool, Ecore Tools, Sirius, and Acceleo. It provides a multi-view representation of an ubiquitous application (see Figure 19).

5.4 Applying Machine Learning

The information collected by the connected objects will be used by a supervised machine learning model to help with prediction. This Module allows you to recognize the different situations based on which decisions must be made. To identify these situations, we propose to use neural networks. Artificial Neural Networks (ANN) work as a mathematical calculation model for data classification. It can provide high-performance reliability in the automatic decision-making process. The multilayer perceptron (MLP) is one of the most used types of ANN.

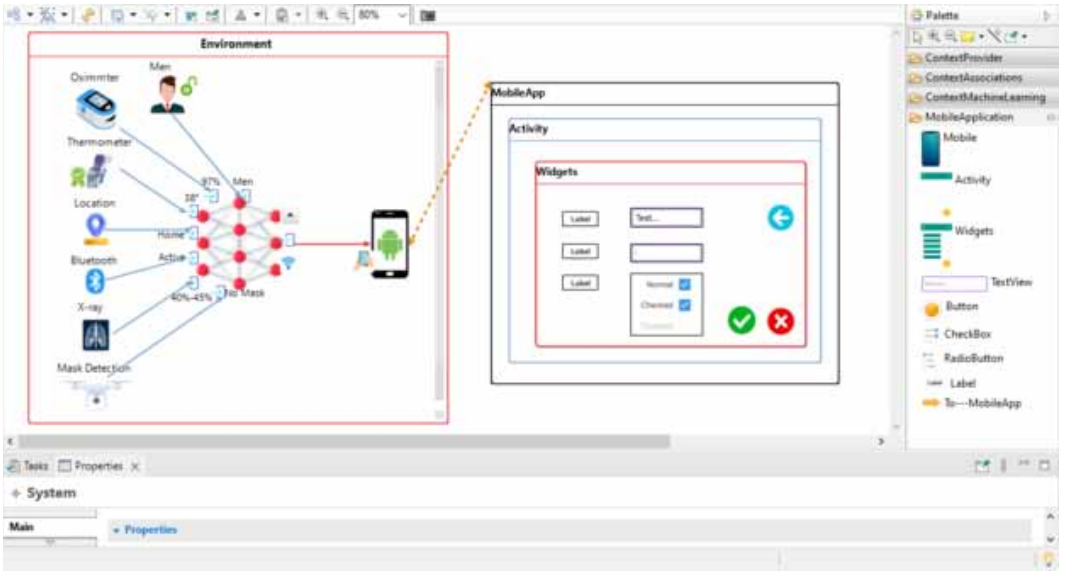
Our work further tries to improve the quality of the classification of COVID-19 patients. It is carried out based on five different symptoms. They are used as input to the multi-layer ANN to categorize and classify patients into two different categories, namely healthy and infectious.

We provide a training dataset, taken from the COVID-19-TRACERSET Public Health France (Sante publique France, 2021). The initial KNN weights are randomly determined and then adapted by the network, and the thresholds (≥ 0.5) predefined in the neuronal cells are applied in the learning process. In supervised learning, when an output is above the threshold, it will be labeled "1", otherwise it will be labeled "0".

Figure 20 represents the proposed architecture of the multilayer MLP composed of two outputs, five input levels, and two hidden layers.

We note (X i) $1 <= i < k$ the k pieces of information arriving from the sensor to the neuron. In addition, each will be valued with respect to the neuron through weight. Weight is simply a coefficient

Figure 19.
Instantiation of the COVID-19 model



Wi linked to the information Xi. The i-th piece of information that will reach the neuron will therefore in fact be W i * X i.

The artificial neuron performs a weighted sum of its inputs rather than considering each piece of information separately. Next, we apply the (sigmoid) activation function to the sum of W i * X i. The activation function is a function that must return either a real close to or a real close to 0. In the training process, each iteration includes the following steps:

- **Feed-Forward:** Calculation of the predicted output y.
- **Back-Propagation:** Updates weights and biases.

Figure 20 illustrates the process of training a neural network model, which includes two steps, namely Feed-forward and Backpropagation.

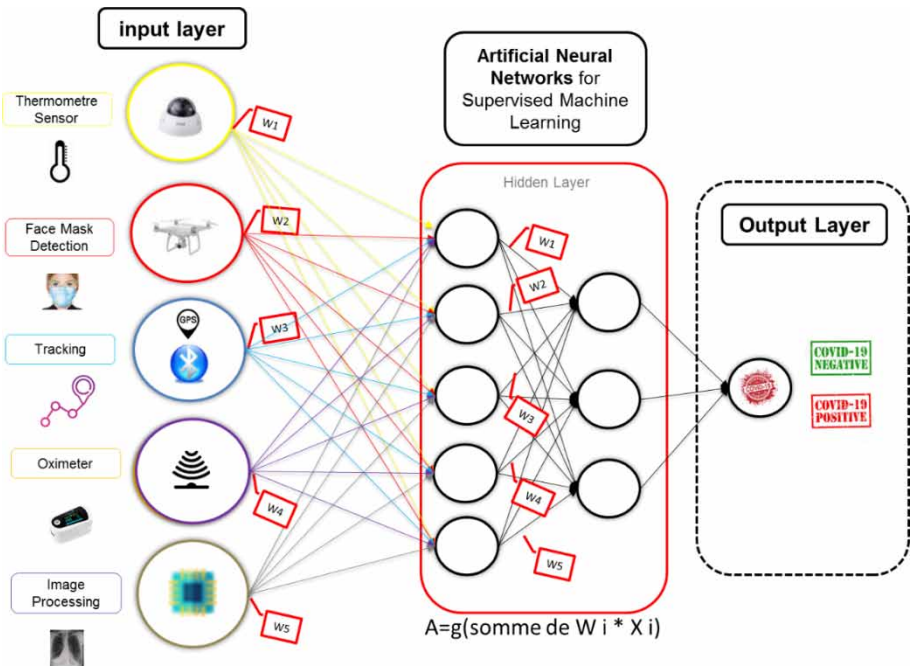
5.5 Create a Ubiquitous Application for COVID-19

We present an illustrative example of our approach to automatically generating the code. The example describes an Android application with three activities and buttons (see Figure 21).

According to the World Health Organization (WHO), the major sign of COVID-19 is temperature. Anyone with a temperature above 37.5 may have a possible infection. Additional signs can also be monitored such as arterial blood oxygen saturation rate, chest X-ray, and incidence rate of the person's geographical position, The system will declare (potential infection) if the following values are recognized by the system: temperature > 38; SpO2 < 94%; Teletorax scanner > 25%, a geographical area with an incidence rate > 200), and the person is not wearing a mask.

We present the final step of the model-driven design, which is the generation of the application code for the Android platform. The figure below illustrates the result obtained from the COVID-19 mobile application, composed of three screens. Each screen contains a set of components. The generator

Figure 20.
Proposed multilayer MLP architecture



developed allows you to generate Android code for this model, but also for any model written with your Android metamodel (see Figure 22).

Patient information is either retrieved via the person's profile, such as age and gender, or received by sensors such as temperature and saturation, or sent by APIs after control examinations such as Scanner X of the thorax. The system sends a notification in the form of a message or suggests that this person do a PCR or an antigen test.

6. EVALUATION SOFTWARE UBIQUITOUS APPLICATION FOR COVID-19

6.1 Dataset Description and Preprocessing

A dataset of confirmed COVID-19 cases from the Public Health France, (Sante publique France, 2021) repository was used. The relevant information extracted from raw data is critical in COVID-19

Figure 21.
Three activity generated from COVID-19 patient app

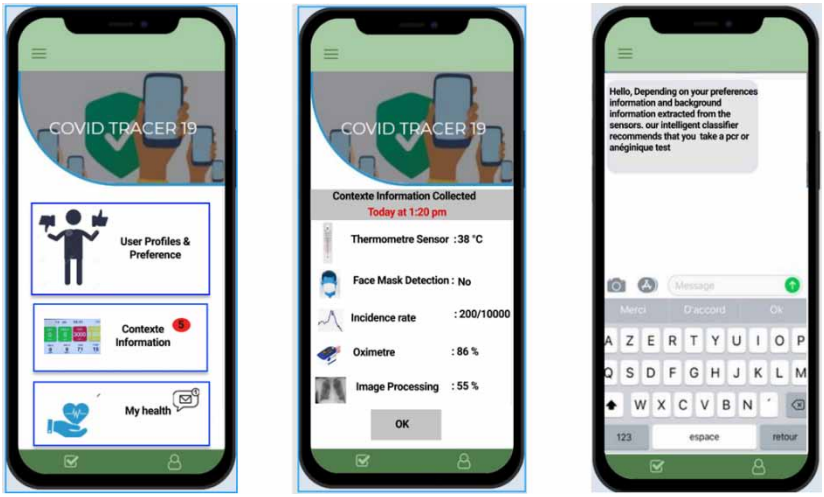
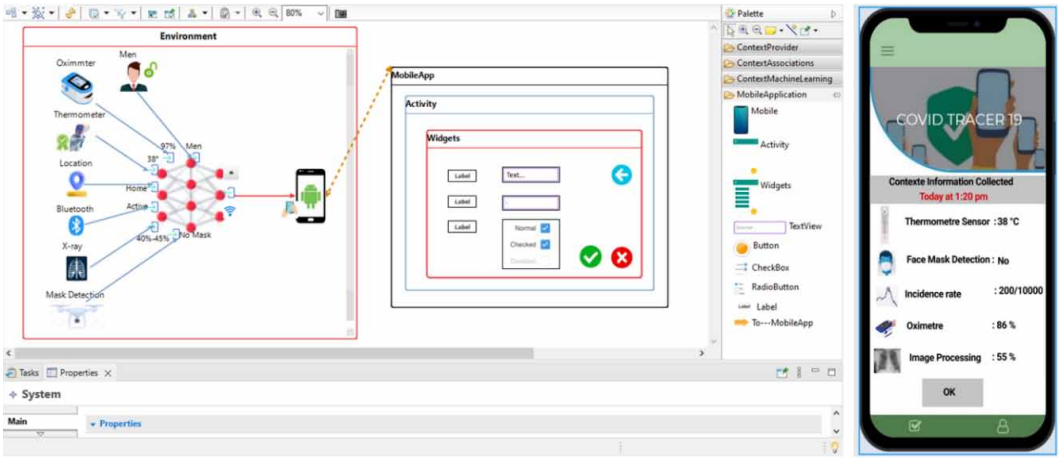


Figure 22.
PIM to PSM



classification due to its direct influence on classification performance. The data contains different types of information about each case, this resulted in a list of 80 symptoms. However, many of these symptoms were judged to be synonyms. Thus, the number of symptoms was reduced to 6. The main input for the classification of COVID-19 cases are arterial blood oxygen saturation rate, chest X-ray, incidence rate of the person's geographical position, temperature and face mask detection.

6.2 Experiments

In our experiments, we employed neural networks, we used multilayer perceptron (MLP) neural networks with 1, 2 hidden layers. After training a model, the accuracy of the trained model is measured on the test set. The inputs for the models are symptoms. The output is positive/negative determination of COVID-19 infection.

6.3 Results and Performance Evaluation Metrics

To evaluate the performance of learning algorithms, two performance measures were used:

6.3.1 Confusion Matrix

The confusion matrix is used to visualize the performance of a binary (2-class) supervised learning problem by creating a 2-by-2 matrix. Each row in the matrix shows the instances in the predicted (or computed) class, while each column shows the instances in the actual class. The resulting matrix consists of four values:

- **True Positive (TP):** Are the number of instances that were classified as positive and are actually positive. It refers to the number of cases that have been accurately identified as Covid-19 patients.
- **False Positive (FP):** Are the number of instances that were classified as positive, but they are actually negative. it stands for the number of cases that have been misidentified as Covid-19 patients.
- **False Negative (FN):** Are the number of instances that were classified as negative, but they are actually positive. it means the number of mistakenly classified instances as not infected, but they are Covid-19, infected patients.
- **True Negative (TN):** Are the number of instances that were classified as negative, and they are actually negative. The number of instances appropriately diagnosed as healthy.

6.3.2 Accuracy

The accuracy of a classifier is computed as the number of correctly classified instances to the total number of instances. A test's accuracy is determined by its capability to differentiate between patient and healthy cases appropriately. It is given by:

$$\text{Accuracy} = (TP+TN)/ (TP+TN+FP+FN)$$

Table 2 shows the accuracy result of the experience with neural network models.

Table 2.
Accuracy result of the experience with neural network

Model	MLP 1	MLP 2
12 symptoms	0,952	0,966
6 symptoms	0,985	0,988

6.3.3 F-Measure

The F-measure is computed by combining the two measures of precision and recall. It is given by:

F-measure= (2*Recall*Precision)/ (Recall + precision)
Recall = TP/ (TP+FN)
Precision = TP/ (TP+FN)

Table 3 shows the precision and recall of the classifiers. Confusion Matrix is a performance measure for machine learning classification.

Figure 23 shows the precision and recall of the 3 classifiers.

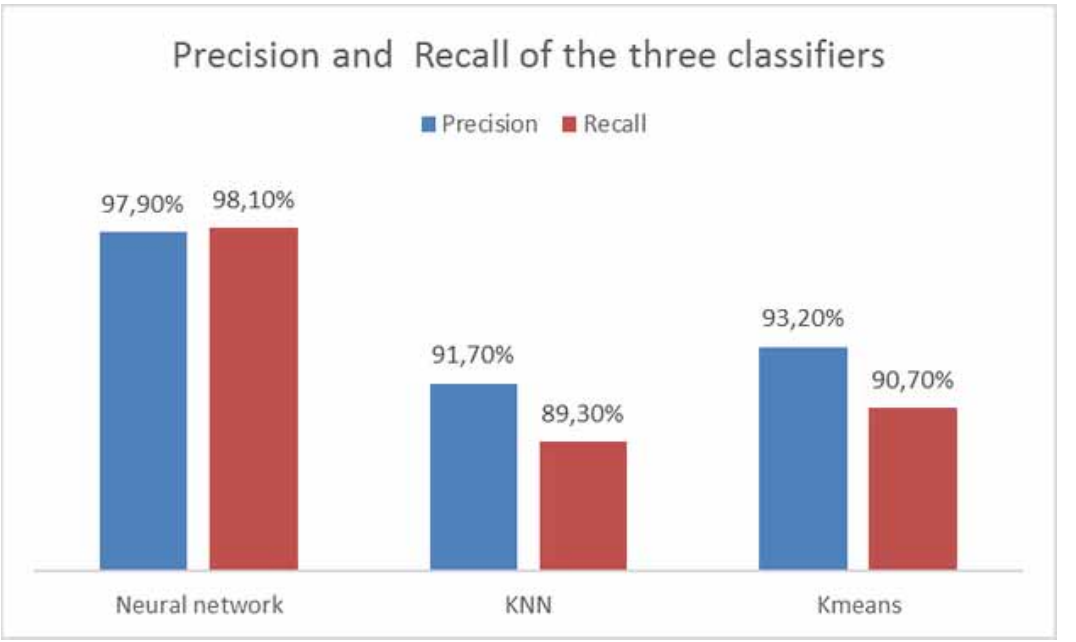
CONCLUSION

This paper proposes a framework (graphics workshop). This editor has been designed for PIM models using the Eclipse Sirius tool. This editor is based on our Meta-model and facilitates the specification of PIM models in a fully graphical way, knowing that the collected measurement COVID-19-Tracer will be used as a training COVID-19-Tracer.

Table 3.
Precision and recall of the classifiers used

Classifies	Neural network	KNN	Kmeans
Precision	97,9%	91,7%	93,2%
Recall	98,1%	89,3%	90,7%

Figure 23.
Precision and recall of the three classifiers used



Currently, we are working on the choice of techniques based on supervised learning. Also, we define the model-to-text transformation (M2T) rules with Acceleo on the example of a COVID-19 Contact-Tracer news application. In order to complete our approach, we are working on the definition of a textual DSL. The objective is to provide a hybrid concrete syntax with a graphic part and a textual part.

We will extend our approach to take into consideration the generation of cross-platform applications (Windows Phone 8, iOS).

REFERENCES

Alballa, N., & Al-Turaiki, I. (2021). Machine learning approaches in COVID-19 diagnosis, mortality, and severity risk prediction: A review. *Inform Med Unlocked.*, *24*, 100564. doi:10.1016/j.imu.2021.100564 PMID:33842685

Baddour, A. M., Sang, J., Hu, H., Akbar, M. A., Loulou, H., Ali, A., & Gulzar, K. (2019). CIM-CSS: A Formal Modeling Approach to Context Identification and Management for Intelligent Context- Sensitive Systems. *IEEE Access : Practical Innovations, Open Solutions*, *7*, 116056–116077. doi:10.1109/ACCESS.2019.2931001

Bedini, F., Maschotta, R., & Zimmermann, A. (2021). A generative Approach for creating Eclipse Sirius Editors for generic Systems. *2021 IEEE International Systems Conference (SysCon),* 1-8. doi:10.1109/SysCon48628.2021.9447062

Boudaa, B., Hammoudi, S., Mebarki, L. A., Bouguessa, A., & Chikh, M. A. (2017). An aspect-oriented modeldriven approach for building adaptable context-aware service-based applications. *Science of Computer Programming*, *136*, 17–42. doi:10.1016/j.scico.2016.08.009

Dalal, K. R. (2020). Analysing the Role of Supervised and Unsupervised Machine Learning in IoT. *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC),* 75-79. doi:10.1109/ICESC48915.2020.9155761

Dorndorfer, J., Hopfensperger, F., & Seel, C. (2019). *The SenSoMod-Modeler – A Model-Driven Architecture Approach for Mobile Context-Aware Business Applications. Information Systems Engineering in Responsible Information Systems* (Vol. 350). Springer.

Duarte, P. A., Barreto, F. M., Gomes, F. A., Viana, W., & Trinta, F. A. (2014). *A Model-Driven Approach to Generate Context-Aware Applications*. WebMedia. doi:10.1145/2664551.2664586

Elhannani, S., Benslimane, S. M., Khalfi, M. F., & Fechfouch, M. (2022). QASIS: A QoC aware stress identification system using machine learning approach. *International Journal of High Performance Systems Architecture*, *11*(1), 12–25. doi:10.1504/IJHPSA.2022.121881

Fedushko, S., & Ustyianovych, T. (2020). Predicting Pupil's Successfulness Factors Using Machine Learning Algorithms and Mathematical Modelling Methods. In Z. Hu, S. Petoukhov, I. Dychka, & M. He (Eds.), *Advances in Computer Science for Engineering and Education II. ICCSEEA 2019. Advances in Intelligent Systems and Computing* (Vol. 938). Springer. doi:10.1007/978-3-030-16621-2_58

Fedushko, S., Ustyianovych, T., & Gregus, M. (2020). Real-Time High-Load Infrastructure Transaction Status Output Prediction Using Operational Intelligence and Big Data Technologies. *Electronics (Basel)*, *9*(4), 668. doi:10.3390/electronics9040668

Hartmann, T., Moawad, A., Fouquet, F., & Le Traon, Y. (2019). The next evolution of MDE: A seamless integration of machine learning into domain modeling. *Software & Systems Modeling*, *18*(2), 1285–1304. doi:10.1007/s10270-017-0600-2

Jaouadi, I., Ben Djemaa, R., & Ben-Abdallah, H. (2018). model-driven development approach for context-aware systems. *Software & Systems Modeling*, *17*(4), 1169–1195. doi:10.1007/s10270-016-0550-0

Khalfi, M., & Benslimane, S. M. (2014). *Systèmes D'information Pervasifs: Architecture et Challenges.* Presented at the UbiMob 14.

Khalfi, M. F., & Benslimane, S. M. (2013). Toward A Generic Infrastructure for Ubiquitous Computing. *International Journal of Advanced Pervasive and Ubiquitous Computing*, *5*(1), 66–85. doi:10.4018/japuc.2013010107

Khalfi, M. F., & Benslimane, S. M. (2015). Meta Model Context Based Space for Ubiquitous Computing. *International Journal of Advanced Pervasive and Ubiquitous Computing*, *7*(2), 51–66. doi:10.4018/IJAPUC.2015040105

Khalfi, M. F., & Benslimane, S. M. (2015). Evaluating Characteristics Adherence Level to Design Framework for Pervasive Projects. *International Journal of Advanced Pervasive and Ubiquitous Computing*, *7*(4), 18–29. doi:10.4018/IJAPUC.2015100103

Khalfi, M. F., Tabbiche, M. N., & Adjoudj, R. (2021). Vers une modélisation graphique des *applications ubiquitaires basée sur un Dsml intelligent: Covid-19 Contact-Tracer*. Colloque sur les Objets et systèmes Connectés - COC'2021, IUT d'Aix-Marseille.

Lalanda, P., Vega, G., Cervantes, H., & Morand, D. (2021). Architecture and pervasive platform for machine learning services in Industry 4.0. *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops),* 293-298. doi:10.1109/PerComWorkshops51409.2021.9431009

Li, X., Tao, X., Song, W., & Dong, K. (2018). AocML: A Domain-Specific Language for Model-Driven Development of Activity-Oriented Context-Aware Applications. *J. Comput. Sci. Technol.*, *33*(5), 900–917. doi:10.1007/s11390-018-1865-9

Mettouris, C., & Papadopoulos, G. A. (2019). *UbiRS4Tourism: Design and Development of Point-of-Interest Recommender Systems Made Easy*. 26th Annual eTourism Conference, Nicosia, Cyprus.

Mohammadi, F. G., Amini, M. H., & Arabnia, H. R. (2020). An Introduction to Advanced Machine Learning: Meta-Learning Algorithms, Applications, and Promises. In M. Amini (Ed.), *Optimization, Learning, and Control for Interdependent Complex Networks. Advances in Intelligent Systems and Computing* (Vol. 1123). Springer., doi:10.1007/978-3-030-34094-0_6

Moin, A. (2021). Data Analytics and Machine Learning Methods, Techniques and Tool for Model-Driven Engineering of Smart IoT Services. *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion),* 287-292. doi:10.1109/ICSE-Companion52605.2021.00130

Moin, A., Rössler, S., & Günnemann, S. (2018). Thingml+: augmenting model-driven software engineering for the internet of things with machine learning. *Proceedings of MODELS 2018 Workshops, co-located with ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018), Copenhagen, Denmark, October, 14, 2018, CEUR-WS.org, CEUR Workshop Proceedings*, 521–523.

Moradi, H., Zamani, B., & Zamanifar, K. (2020). CaaSSET: A framework for model-driven development of context as a service. *Future Generation Computer Systems*, *105*, 61–95. doi:10.1016/j.future.2019.11.028

Nadjib Tabbiche, M., Fethi Khalfi, M., & Adjoudj, R. (2021). A Smart Modeling Tool for Model-Driven Engineering of Ubiquitous Applications: Covid-19 Contact-Tracer. *2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI),* 1-6. doi:10.1109/ICRAMI52622.2021.9585950

Ozkaya, M., & Akdur, D. (2021). What do practitioners expect from the meta-modeling tools? A survey. *J. Comput. Lang.*, *63*, 101030. doi:10.1016/j.cola.2021.101030

Rademacher, F., Sorgalla, J., Wizenty, P., Sachweh, S., & Zündorf, A. (2020). Graphical and Textual Model-Driven Microservice Development. In Microservices. Springer. doi:10.1007/978-3-030-31646-4_7

Ruiz, J., Serral, E., & Snoeck, M. (2019). *Evaluating user interface generation approaches: model-based versus model-driven development*. Academic Press.

Santé Publique France. (2021). *Données hospitalières relatives à l'épidémie de COVID-19*. https://www.Covid-19-Tracer.gouv.fr/fr/Covid-19-Tracersets/donnees-hospitalieres-relatives-a-lepidemie-de-covid-19/

Serafettin, S., & Ya¸sark, H. S. (2019). Model driven security in a mobile banking application context. *14th International Conference on Availability. Reliability and Security*, 1–7.

Sheng, Q.Z., Yu, J., Xu, H., Zhang, W., Ngu, A.H., Han, J., & Liu, R. (2018). *ContextServ: Towrds Model-Driven Development of Context-AwareWeb Services*. ArXiv, abs/1811.12573.

Trajano, I. A., Ferreira Filho, J. B., de Carvalho Sousa, F. R., Litchfield, I., & Weber, P. (2021, February). MedPath: A process-based modeling language for designing care pathways. *International Journal of Medical Informatics*, *146*, 104328. doi:10.1016/j.ijmedinf.2020.104328 PMID:33281069

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, *265*(3), 66–75. doi:10.1038/scientificamerican0991-94