Integration of ubiquitous specifications in the conception of objects system

Sonia Aimene* and Idir Rassoul

LARI Laboratory of Computer Science,
Mouloud Mammeri University of Tizi-Ouzou,
Tizi-Ouzou 15000, Algeria
Email: aimene.sonya@gmail.com
Email: idir_rassoul@yahoo.fr
*Corresponding author

Abstract: This paper proposes an approach called Ubiquitous System Object (Ubi-SO) based on standard life cycle. This approach aims to identify, to analyse and to design ubiquitous requirements that can be incorporated into a traditional system engineering process. The approach is modelled with the Business Process Management Notation (BPMN) method which is adapted using the Bizagi Modeler tool. Ubi-SO separates functional, technical and contextual ubiquitous needs in conceptualisation phase. It is based on extended sequence diagram in analysis phase and on extended class diagram in design phase using Uml profile for the best adaptation to ubiquitous domain. Compared with a lot of works, this solution offers guidelines for studying and spreading ubiquitous needs. To demonstrate the feasibility of our work, the approach is verified by translating the BPMN into formal Language of Temporal Ordering Specification (LOTOS) and then validated by the CADP tool.

Keywords: modelling; context-awareness; ubiquitous computing; pervasive applications; mobility; LOTOS; BPMN; Uml profile; life cycle.

Reference to this paper should be made as follows: Aimene, S. and Rassoul, I. (2022) 'Integration of ubiquitous specifications in the conception of objects system', *Int. J. Computer Applications in Technology*, Vol. 68, No. 1, pp.70–81.

Biographical notes: Sonia Aimene is a PhD candidate in the Computer Science Department of the Electrical Engineering and Computer Science Faculty at the Mouloud Mammeri University of Tizi-Ouzou, Algeria. She holds her MSc degree in Computer Systems and Networks from the same department in 2014. Her research interests include pervasive information systems, integration of ubiquitous specifications in the conception of object systems and information systems.

Idir Rassoul joined the Mouloud Mammeri University of Tizi-Ouzou, Algeria, where he works as University Lecturer empowered to conduct research in Computer Sciences. After his studies at the University of Paris IX Dauphine (France) where he obtained the Doctorate 3rd Cycle in Computer Science, Head of a Research Team at the Laboratory of Research in Computer Science (Lari) in the field of Software Engineering and Information Systems. Currently, he is developing Collaboration in Geographic Information Systems including Pervasive Computing and Decision-Making in Emergency Situations in Collaboration with colleagues of Department of Civil Engineering.

1 Introduction

Extended object oriented development systems offer promising and advantageous perspectives to ubiquitous system. Today, the ubiquitous computing paradigm aims at allowing a smooth integration of the information technologies in diverse domains including education and healthcare (Ahmed et al., 2017; Escobar et al., 2017; Batarseh and Gonzalez, 2018; Shahzad et al., 2020). So, as computing systems become more pervasive and complex, the nature of applications must change accordingly. Object oriented systems must then adapt and support this new vision, which is ubiquity. They must also allow the orchestration of services in an ubiquitous environment. Pervasive ubiquitous computing is distinguished from traditional computing systems by its integration to pervasive environment and adaptation to the context (Ferscha, 2011). In order to ensure the usability and ubiquity of applications, it is necessary to meet the requirements of users. The literature does not specifically address the engineering of ubiquitous requirements and does not provide methods to facilitate their specification and so, only a few approaches provide engineering methods based on a detailed development process. For example, there are no guidelines for modelling in Henricksen and Indulska (2006) and there is no engineering requirements in Vieira et al. (2011) and Cipriani et al. (2011). An ubiquitous model cannot be complete without adopting a

development methodology to organise the engineer's work to have a process that can facilitate the design of ubiquitous applications. The literature addresses these challenges primarily related to the development of processes for capturing, acquiring and modelling contextual information. To address this challenge, this paper develops an approach for the conceptualisation, analysis and design of object-oriented systems that follow the traditional steps by proposing a new process called Ubiquitous System Object (Ubi-SO).

This process is based on standard life cycle (Booch, 1995) and allows identifying, analysing and designing ubiquitous requirements that can be integrated into traditional system engineering process. The objective of this approach is to facilitate the engineering of pervasive applications by integrating ubiquitous aspect that is context-awareness which is an important aspect in pervasive computing. This will offer a solution that provides guidelines for studying and modelling an ubiquitous process. We model our approach with the BPMN (White, 2004) method which can be adapted with the Bizagi Modeller tool for better management and modelling of the process using its Build time environment in the analysis and designing parts. The characteristics of our approach are:

- The reuse of phases of a classical system which allows to benefit and to exploit the bases of engineering.
- Conceptualisation step proposes separation of logical, technical and contextual ubiquitous specifications to allow easy modification and better maintenance.
- These specifications will be analysed in the 'analysis' step where we propose the extension of the sequence diagram using the UML profile and they will be designed in the 'design' part where we also proposed the extension of the class diagram with the UML profile.

The proposed UML ubiquitous profile in analysis and design steps is a package of specific profiles that extend the standard notations of two UML diagrams chosen according to different views of a system (sequence diagram and class diagram). For each diagram, we propose UML extension mechanisms such as stereotypes, constraints and tagged values that can model any contextual situation. The UML profile is used because it allows, thanks to its customisation, an adaptation of the situations to the contextual environment. We will then have all the management steps of this ubiquitous process starting from the identification of needs to their design and verification.

The approach is verified using LOTOS (Ameyed, 2017) specification, where we translate each activity of process of sequence diagram to formal language and validate them with CADP tool. LOTOS is based on process algebra and it is extremely useful for temporal logic verification and for simulation. The formal specification language LOTOS was chosen because it is an ISO (International Organisation for Standardisation) standard for which several verification tools such as CADP are available. This verification is performed through the proof of behavioural properties related to the composition scenario, expressed using temporal logic. The rest of this paper is organised as follows: Section 2 discusses related work concerning the development of ubiquitous Information Systems (ISs), basic concepts and foundations for the approach are presented in Section 3, Section 4 gives the case study, Section 5 verifies Ubi-SO formally and validates LOTOS programs, and finally, Section 6 concludes the paper and describes the further works.

2 Related work

More than 25 years ago, Mark Weiser identified ubiquitous computing as the third generation of computers, after the mainframe generation and the personal computer generation (Weiser, 1991). This vision reflects how we can define current and future generations of computers based on technological innovations and applications that influence the human experience in computing (Abowd, 2016). There are different studies and researches in ubiquitous domain which has many modelling methods. Current approaches to modelling and context management can be found in Bruneliere et al. (2019). Wang et al. (2018) aimed at providing a concise overview of the technical characteristics and applications of ubiquitous manufacturing systems which were published between 1997 and 2017. Various approaches to modelling ubiquitous systems are based on model-oriented approaches like approach of Henricksen and Indulska (2006); Vieira et al. (2011); Cipriani et al. (2011) and Ameyed (2017). A set of conceptual models designed to support the software engineering process is proposed in Henricksen and Indulska (2006), including context modelling techniques, two programming models and a preference model for representing context-dependent requirements. They also present a software engineering process and software infrastructure that can be used in conjunction with their models. Vieira et al. (2011) presented an integrated approach to assist the design of Context Sensitive Systems (CSS). The originality of this work lies in the proposed way of thinking about context, on the proposed context metamodel and on the specification of a process for designing CSS. The meta-model supports building context models by making explicit the concepts related to context manipulation and by separating the context structure model from the CSS behaviour model. The design process details the main activities related to context specification and the design of CSS, providing a systematic way to execute these tasks. To provide a graphical user interface to design schemas for spatial and technical context models in Cipriani et al. (2011) presentd the NexusEditor which interactively creates queries, send them to a server and visualise the results. One main contribution is to show how schema awareness can improve such a tool. The NexusEditor dynamically parses the underlying data model and provides additional syntactic checks, semantic checks and short-cuts based on the schema information. Furthermore, the tool helps to design new schema definitions based on the existing ones, which is crucial for an iterative and user-centric development of context-aware applications. Ameyed (2017) presented a new formal approach to context prediction in context-sensitive proactive systems. They express context and transition in a diffuse system under a formal presentation, using probabilistic temporal logic PCTL (a probabilistic extension of temporal

logic). They propose a probabilistic transition model to code the behaviour of the system over time. The combination of PCTL with the stochastic model allows us to plot, analyse and predict the future context using the checking model to verify the properties of future states and thus returning quantitative results. Most of the related works focus on just the contextual aspect but neglect the functional and technical ones. We deduce that most approaches offer context meta-models, but do not provide guides or instructions for using them to facilitate the task of designers. For example there is no guideline for adding requirement context modelling analysis and specifications phase in Henricksen and Indulska (2006) and there is no requirement engineering in Vieira et al. (2011) and Cipriani et al. (2011). In this paper, we aim to consider several criteria by proposing an approach that includes guide or instructions to the engineer and make it easier to handle the requirements of his system. The approach separates functional, technical and contextual ubiquitous needs by proposing ubiquitous specification process. These specifications will be analysed in the 'analysis' step where we propose the extension of the sequence diagram using the UML profile and so, they will be designed in the 'design' part where we also proposed the extension of the class diagram with the UML profile. This solution offers guidelines for studying and spreading ubiquitous needs.

3 Proposed approach

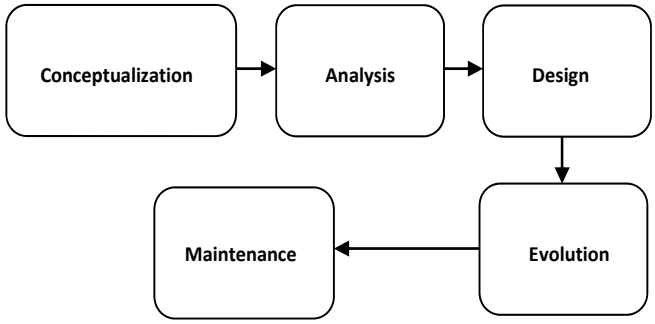
This section is devised in three main parts. In the first part, we give general concepts of standard life cycle which is the basis of our new proposal. Then, in the second part, we explain how the ubiquitous specifications are integrated. Finally, we show modelling of these specifications using Build Time environment. After this, their analysis and design are given using UML Profile.

3.1 Basic concepts

Our approach is inspired by the classical standard life cycle which includes all stages of the development of any process depicted in Figure 1. This life cycle is composed of 5 steps which are described below.

Conceptualisation step seeks to establish an overview of an idea and validates its basis which is necessary to define all the objectives in the concepts. Then, analysis step allows modelling by identifying the classes and objects (their roles, responsibilities and collaborations) that form the vocabulary of the problem area. To create architecture for implementation and allow creating diagrams, the designer uses the design phase. Finally, the evolution and maintenance phases respectively allow the implementation to be developed and modified through successive refinements by making updates, in order to obtain the final system for delivering the designed product.

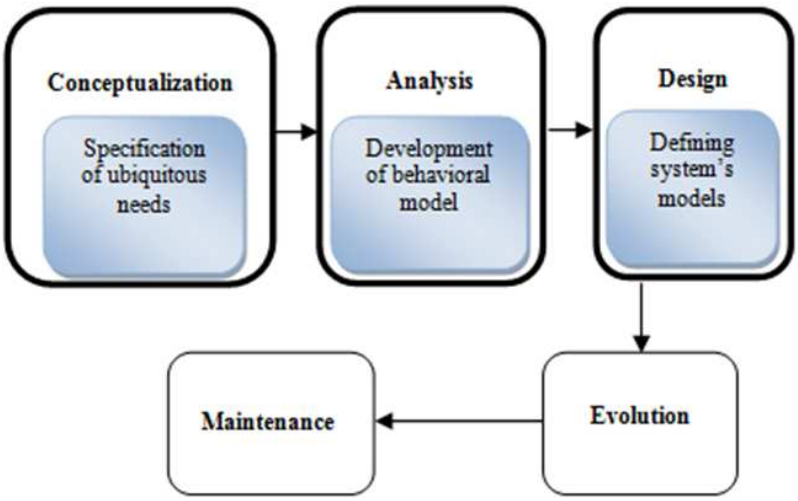
Figure 1 Life cycle of specification and design



3.2 Integration of ubiquity

Ubi-SO life cycle proposed in this paper consists of integrating the ubiquitous specifications in standard object oriented life cycle as follows (see Figure 2).

Figure 2 Ubi_SO life cycle

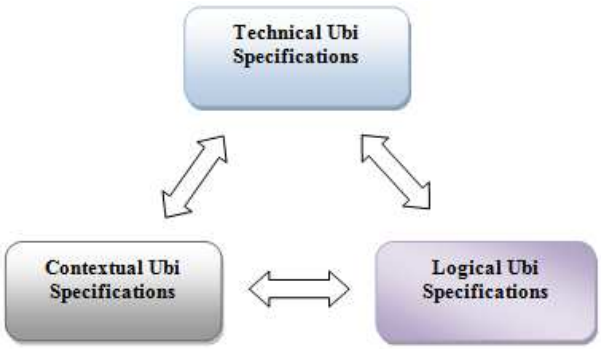


3.2.1 Conceptualisation

Specification of ubiquitous needs: It is the acquisition of needs and it consists of defining the software, the hardware and the contextual needs according to a set of requirements. This step allows the designer to detail and classify his material needs such as the use of different equipment's. He must specify also his software and his contextual needs such as the study of the working environment which includes contextual information. In this step, we have developed ubiquitous specifications process which is composed of three parts including technical Ubi (Ubiquitous), logical Ubi and contextual Ubi specifications (see Figure 3). To better see the user's needs, we have separated the logical, contextual and technical specifications. The technical specifications represent the choice of hardware. The logical specifications represent the choice of software. The contextual specifications define the contextual data and their integration into the existing environment (See Figure3).

- *Logical Ubi specifications*: This step represents the user as an entity that can manage and control his needs with his ubiquitous equipments. It defines also the interaction between users.
- *Contextual Ubi specifications*: This step includes all the constraints related to the used context in system or application. It consists of collecting and capturing all the elements that can influence the current situation of a user in an environment.
- *Technical Ubi specifications*: It represents the data source and equipments including data detection sensors and surveillance cameras.

Figure 3 Ubiquitous specifications process



3.2.2 Analysis

Development of behavioural model: The definition of system's behaviour is added to analysis phase. In this step an analyst defines the interaction with users and models contextual data using ubiquitous sequence diagram. This later is obtained with the extension of basic sequence diagram using concepts of Uml Profile.

3.2.3 Design

Defining system's models: In this step, we have defined the models of system. We propose a ubiquitous class model that models contextual information and includes all the characteristics that intervene in the ubiquitous environment. Ubiquitous class diagram is obtained with the extension of basic class diagram using concepts of Uml profile.

3.2.4 Evolution

This step allows the engineer to obtain a flexible system that can be quickly modified in case of new constraints or strategies, while taking into account the adaptations and customisations suggested by the work team.

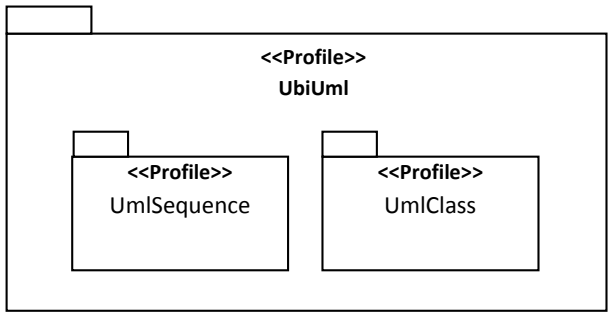
3.2.5 Maintenance

This step allows all the updates and modifications of a system while leaving its basic functionalities intact. It thus allows rectifying the anomalies and bringing the suggested improvements. There are several levels of maintenance, like preventive maintenance, improving maintenance and corrective maintenance.

3.3 Ubi-SO modelling approach

In this paper, Built Time environment of BPMN (White, 2004) and concepts of Uml Profile are used for modelling, analysing and designing our process. The techniques and methods of BPMN allow identifying and modifying existing processes to align them with a desired future state. Previously, we have seen that there is a possibility to separate ubiquitous contextual specifications from logical and technical ones. For analysing and designing these specifications, we have opted to use the Uml profile which is a generic extension mechanism for Uml. It can change the representation of Uml concepts and can add constraints to these concepts as well as constraints on the use of these concepts. Figure 4 represents the global package of the proposed Uml profile which is composed of two packages and each of them represents Uml profile. This explains the global package proposed for the analysis and design part which is called UbiUml profile (Ubiquitous Uml profile). In this article, we customised the sequence diagram and the class diagram. The sequence diagram in the analysis part means the study of the system behaviour. System behaviour is the exchange of interaction and communication between objects in the environment. The choice of the class diagram in the design part is due to the fact that it is representative of the different diagrams of the system.

Figure 4 Package of the proposed profile



Sequence Uml profile and Class Uml profile extend the notion of sequence and class diagram to support ubiquitous characteristics. Both profiles belong to the meta-class package of Uml.

3.3.1 Analysis

The purpose of the analysis part is to follow the system's behaviour. In this article we have used the sequence diagram which allows showing the interactions and scenarios between the objects of the system. In order to make the sequence diagram suitable for ubiquitous design, we have extended it using the Uml profile. The goal is to build Uml profile composed of stereotypes, constraints and attributes that can adapt Uml concepts to ubiquitous concepts. A stereotype allows defining new semantics by adding it to an existing element of the Uml meta-class. Constraints define the conditions applied to the new elements. Figure 5 shows the proposed stereotype which is the extended sequence diagram explained below:

- 'UbiObject' represents ubiquitous entity in interactive system.
- 'UbiLifeLine' represents the period during which there will be different interactions between 'UbiObject'.
- 'UbiMessage' concerns messages exchanged in the environment between two 'UbiLifeLine' corresponding to two 'UbiObject'.

In the following, we describe how we extend the Uml sequence profile including package, constraints and attributes profiles.

A *Package profile*: Figure 6 shows the extension of the 'Object', 'LifeLine' and 'Message' metaclasses using the 'UbiObject', 'UbiLifeLine' and 'UbiMessage' stereotypes. Each of these concepts will be able to model in the ubiquitous domain, allowing accurate modelling of ubiquitous specifications. The meta-class Object is an instance of the meta-class Class and each stereotype is inherited from its basic meta-class.

B *Constraints profile*: The proposed stereotype must comply with the following conditions:

- 'UbiObject' can interact with another 'UbiObject' using 'UbiMessage'.
- 'UbiLifeLine' highlights chronologically the sending and receiving points of the stereotype 'UbiMessage'.
- 'UbiMessage' must have the 'UbiLifeLine' of the source 'UbiObject' as the starting point and the 'UbiLifeLine' of the target 'UbiObject' as the end point.

C *Attributes profile*: Attributes allow appearing the change in using of context in the ubiquitous domain. 'UbiObject' can have a name, a role and a type. UbiMessage can be asynchronous or synchronous, (see Figure 7). Figure 8 illustrates Graphical representation of exchanged messages between objects in sequence diagram.

Figure 5 Extended sequence diagram

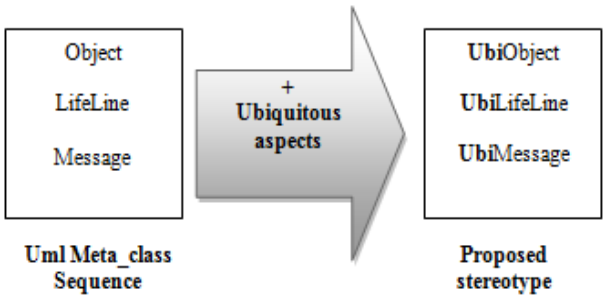


Figure 6 Proposed package profile

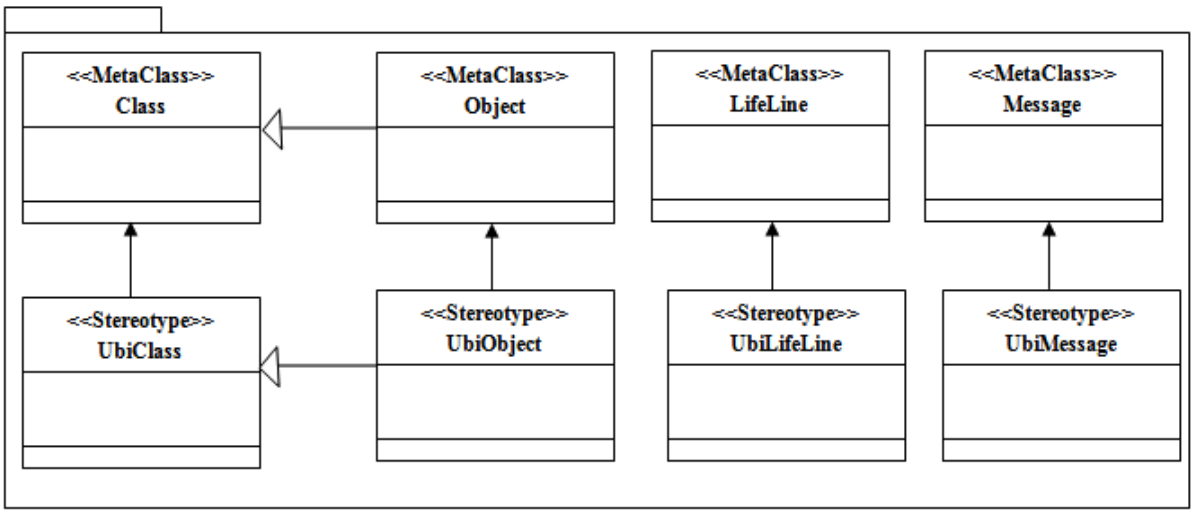


Figure 7 Profile attributes

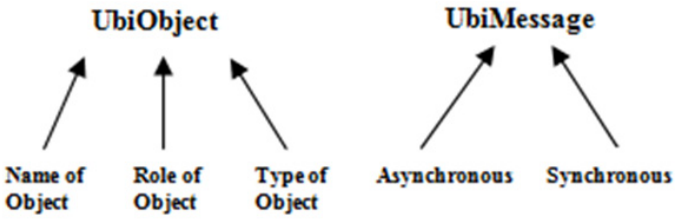
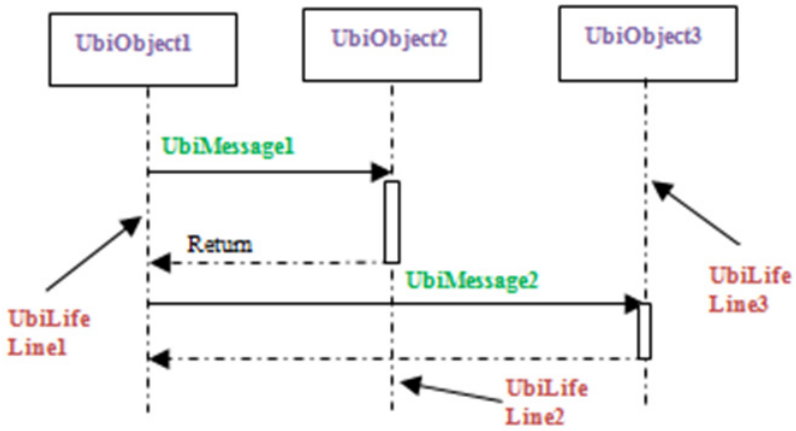


Figure 8 Graphical representation of exchanged messages between objects in sequence diagram

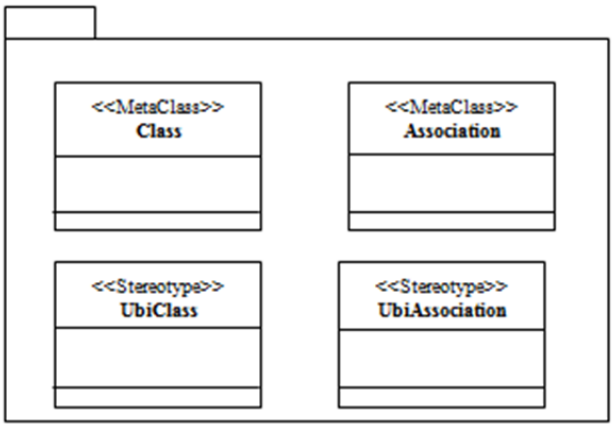


3.3.2 Design

In the design part, we have chosen to represent and extend the class diagram.

A *Package profile*: Figure 9 shows the extended Uml Class profile package. 'UbiClass' is an extension of the Class element and 'UbiAssociation' is an extension of the association element.

Figure 9 Proposed package



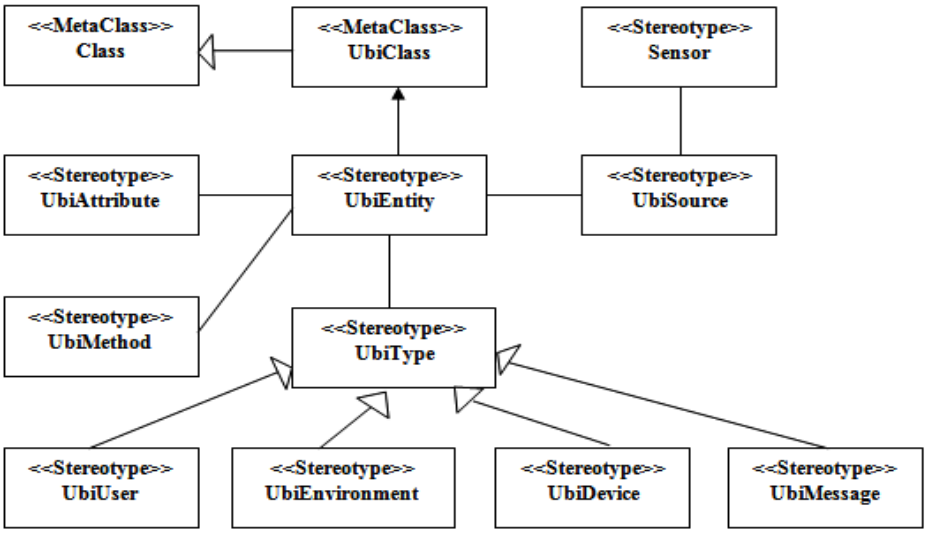
B *Constraints profile*: Each ubiquitous class will have constraints and attributes like the base classes. Each class is any entity of the system that we called 'UbiEntity' which defines the ubiquitous elements of the system. These contextual elements can be the user, the environment, the devices used in the environment and the messages exchanged between entities. We have represented the user entity by the stereotype 'UbiUser' that can receive and transmit information. The 'UbiEnvironment' stereotype represents the environment of the system where contextual information circulates. The 'UbiDevice' stereotype represents any device used in the environment and it can also be applications. The 'UbiMessage' stereotype concerns the messages exchanged in the environment.

Figure 10 shows the proposed stereotypes for the ubiquitous meta-class 'UbiClass'. The extension of the Class meta-class is the 'UbiClass' stereotype which extends 'UbiEntity' stereotype. 'UbiEntity' has as type 'UbiType' which has four types of stereotypes. UbiUser, UbiEnvironment, UbiDevice and UbiMessage have as constraint their association with 'UbiType' by generalised association. The 'Sensor' stereotype is associated to the 'UbiSource' stereotype by simple association. The two other stereotypes 'UbiAttribute' and 'UbiMethod' are attached to the 'UbiEntity' stereotype by simple association.

C *Attributes profile*: Each stereotype must have its own attributes and methods. Example for the case of UbiUser can be static by having only attributes or dynamic if it changes location and UbiMessage like text, videos, etc.

In the next section, we evaluate our approach using case study and in Section 5, we verify and validates it using LOTOS specification with CADP tool.

Figure 10 Proposed stereotypes for 'UbiClass'



4 Case study: healthcare environment

The approach proposed in this paper is evaluated using healthcare application as case study illustrated by scenario of validation (see Figure 12) modelled by Bizagi modeler. Bizagi Modeller is used as software for modelling processes, especially in BPMN format. The application consists of validating scenario to obtain results in the form of ubiquitous requirements.

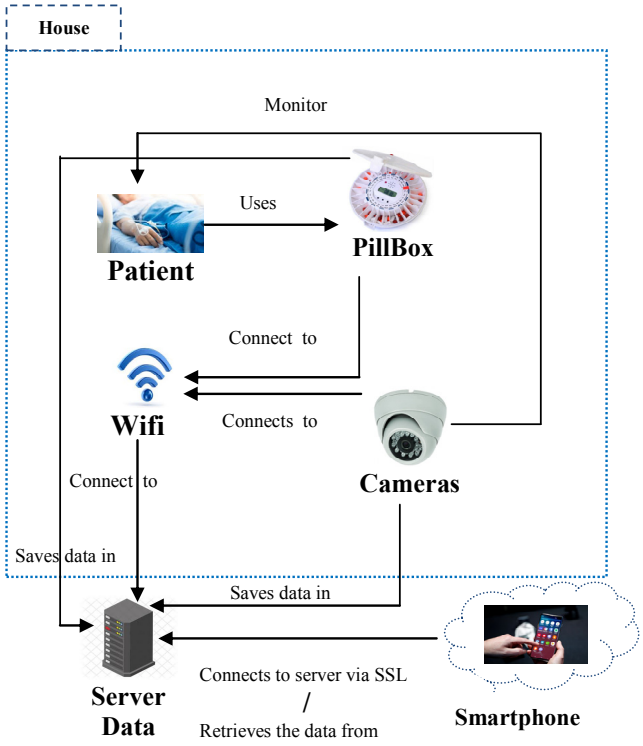
4.1 Scenario

Mary has a diabetic mother with a physical handicap which led her to provide a PillBox, in order to take her medication by herself when Mary is away. An application that interacts with the surveillance cameras and the PillBox is installed on Mary's phone. During the installation of this last one we will ask for the serial numbers of the two devices (PillBox and Camera) for synchronisation and a connection to server. The PillBox and the cameras are connected to the Wi-Fi network of the house. In case the patient forgot to take her medication, Mary will receive a negative message from the PillBox on her Smartphone. She will then trigger the PillBox alarm and if the PillBox still sends negative messages, she will be forced to check on her mom with the surveillance cameras. The video data from the cameras, the PillBox data and the patient's data are recorded on a server with a network card that is also connected to the application.

4.2 Conceptualisation

Figure 11 shows technical ubiquitous specifications which include all used materials in this project of validation. In this case study, the Smartphone is used to control a patient remotely, which allows to trigger a surveillance camera to visualise the patient's conditions. The Pillbox provides the patient with medication. The Server contains patient's information and different data. Logical specifications used in application are android application for Smartphone. We conclude also that the ubiquitous needs have been clearly separated during this application as contextual needs. For example, the Pillbox alarm is a data that has the Pillbox as resource. This data is considered as logical need. SMS sent by Mary which is data that circulate through Wi-Fi and the video images obtained from the surveillance camera triggered by the Smartphone. All these data are considered as contextual informations which circulate in healthcare environment.

Figure 11 Technical ubiquitous specifications in healthcare environment



4.3 Analysis

The objective is to describe the simulation of different scenarios of the process described in Figure 12. At the beginning of the patient control, we check whether we have

a negative or positive message. If it is positive, we go to the final process and if not, we ask the PillBox to activate the alarm. Activating the alarm allows to remind the patient to take medication. Then, we check if the patient is taking his medication. If we receive a positive message we complete the process. If not, we activate the monitoring cameras to check the patient's conditions.

4.4 Design

In the design part of our case study, we have developed the class diagram (see Figure 13) concerning the executed scenario. This diagram uses the notation extensions elaborated in Figures 9 and 10. The stereotypes represent the specific classes of the case study and use the ubiquitous classes. Healthcare is the ubiquitous class and the classes 'UbiUser', 'UbiEnvironment', 'UbiDevice' and 'UbiMessage' are inherited from this class. 'UbiSource' which is the location class has as class 'Home' and 'work' which are user locations in the case study. After illustrating our case study with BPMN model of sequence diagram in analysis phase, we propose in the next section to verify it with formal specification, because, sequence diagram with BPMN format does not provide functionality for formal verification and this is due to the lack of formalism. They are used generally for graphical specification.

Figure 12 Sequence diagram of remote patient monitoring using BPMN format

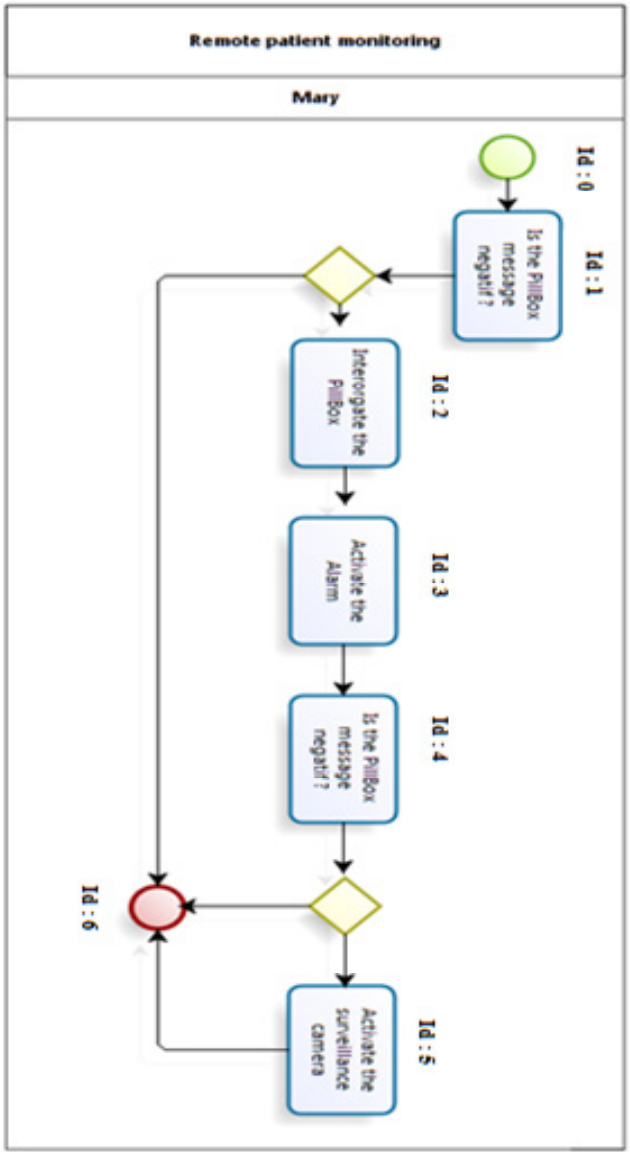
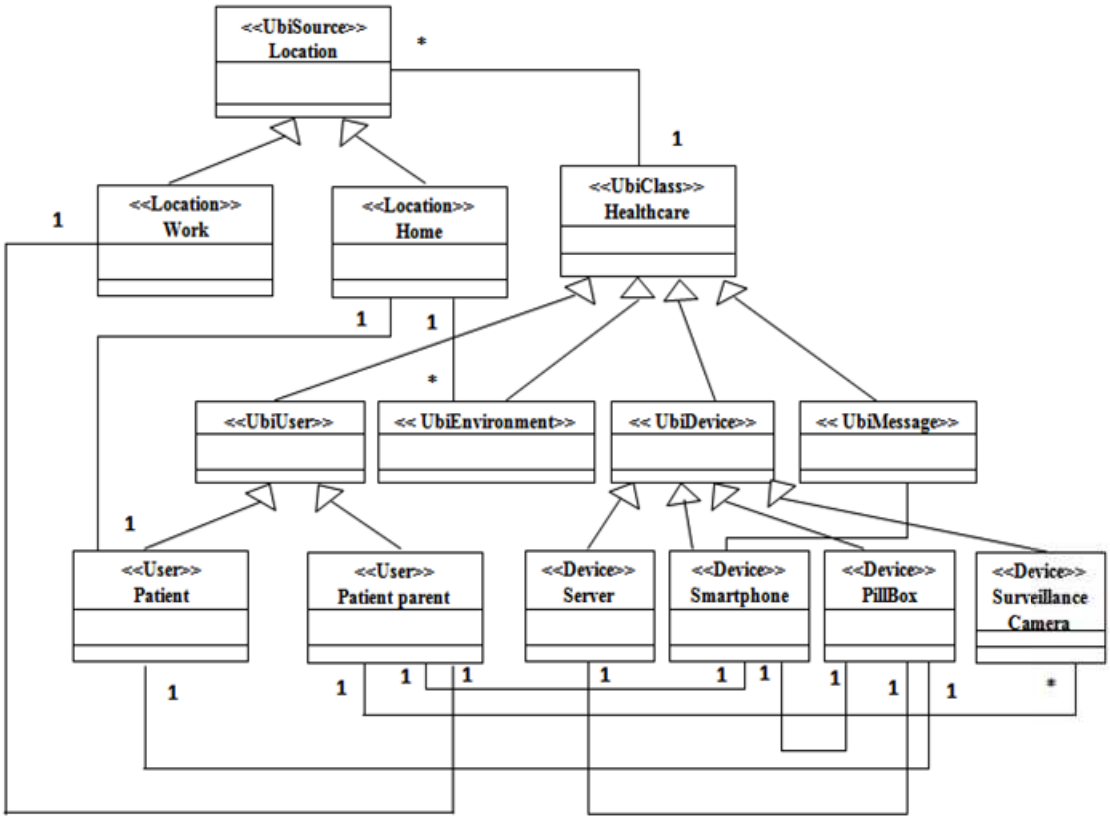


Figure 13 Case study class diagram



5 Formal verification using LOTOS

Formal verification is the systematic process of verifying through algorithmic techniques that an implementation is in accordance with the specification and all execution paths are analysed mathematically. A lot of works have been done in this domain like contribution presented in Dos Santos et al. (2019).

The process algebra LOTOS is formal description language for specifying competitive systems (Brinksma, 1988). A Lot of authors have used it, e.g. Dumez (2010) used UML activity diagram to model the composition of web services and they transformed it into LOTOS specification.

Verification is an essential step in any development approach and therefore in our approach also. We have chosen the formal verification because we consider that it is more reliable and requires less work from the developer since the formal specification of the system is automatically generated. The developer does not need to perform test sets or to proceed to the simulation of the system execution.

5.1 An overview of LOTOS Language

LOTOS is based on temporal ordering of events and process algebraic methods (Dumez, 2010) and uses various behaviour operators. These are summarised in, Figure 14 where *G* refers to a gate (channel of communication), *x* to variable, *P* to process, S to sort, v to value and B to behaviour.

Figure 14 LOTOS behaviour operators

Behaviour Operator	Meaning
stop	inaction
G !V ?X:S ; B	Action Prefix
B1 [] B2	Choice
[E] -> B	Conditional
B1 ||[G1,...,Gn]|| B2	Parallel composition
B1 ||| B2	Interleaving
exit	Successful termination
B1 _ B2	Sequential composition
P [G1,...,Gn] (V1,...,Vm)	Process call

5.2 LOTOS specification for healthcare environment

To translate the BPMN environment depicted in Figure 12 into LOTOS, we have followed different steps.

1) Define a process for each step of the activity including the start and the end node. In our case the processes are:
   - To control the patient
   - To interrogate the electronic box

- To check patient's conditions
- To monitor patient with camera

2) Assign an identifier to each process (id) of type integer for a better follow-up.
3) Define the communication channels between processes. A process is in between (SEND and REC). A service can send or receive a message (event) via the SEND and RECV channels.
4) Define operations between processes. In our case we use the operator ||| which means that the processes are independent and we also use the notation |[SENDi , RECVi]| to synchronise the processes of the service with the BUSi where 'i' is between 0 and N. We Identify control-flow patterns in the workflow in order to define (implement) each process. In the beginning we translate the sequence diagram into LOTOS.

Algorithm 1 represents the instantiation in LOTOS of this process.

Algorithm 1: Processes instantiations in LOTOS

```
specification PatientRemoteControl [SEND, RECV]:
noexit
behaviour
(
Init [SEND, RECV](0)
|||
PatientControl [[SEND, RECV] (1)
|||
InterogatePillBox [SEND, RECV] (2)
|||
ActivateAlarm [SEND, RECV](3)
|||
CheckPatientCondition [SEND, RECV] (4)
|||
StartSurveillanceCamera [SEND, RECV](5)
|||
Final [SEND, RECV](6)
)
|[SEND, RECV]|
BUS [SEND, RECV] (<>)
where
(*Processes definition*)
   endspec
```

In the next stage, we will identify the control-flow in the workflow in order to provide implementation for each process. Init process (id: 0) starts the PatientControl process (id: 1) as a consequence, it uses the sequence pattern as defined in Algorithm 2.

Algorithm 2: LOTOS specification for Init process

```
process Init [SEND, RECV] (Id:Int): exit :=
 Sequence [SEND, RECV] (Id, 1)
 >> exit
   endproc
```

The PatientControl process waits for a run message from Init process before starting. After that, it realises an exclusive choice between InterogatePillBox process (id: 2) and the Final process (id: 6) as defined in Algorithm 3.

Algorithm 3: LOTOS specification for PatientControl process

```
process PatientControl [SEND, RECV] (Id:Int) : exit:=
 RECV !Id !0 !RUN ! void;
 ExclusiveChoice [SEND , RECV] (Id ,
 insert (6, insert (2, {} ) ))
 >> exit
   endproc
```

The InterogatePillBox process waits for a run message from the PatientControl process before starting the ActivateAlarm (id: 3), this realises a sequence pattern. The corresponding specification is provided in Algorithm 4.

Algorithm 4: LOTOS specification for InterogatePillBox process

```
process InterogatePillBox [SEND, RECV] (Id:Int):
exit:=
RECV !Id !1 !RUN ! void;
Sequence [SEND, RECV] (Id, 3)
>> exit
   endproc
```

The ActivateAlarm process waits for a run message from the InterogatePillBox process before starting. The ChekPatientCondition (id: 4), this realises a sequence pattern, as defined in Algorithm 5.

Algorithm 5: LOTOS specification for ActivateAlarm process

```
process ActivateAlarm [SEND, RECV] (Id:Int):
exit :=
RECV !Id !2 !RUN ! void;
Sequence [SEND, RECV] (Id, 4)
>> exit
endproc
```

The CheckPatientCondition process waits for a run message from the ActivateAlarm process before starting the exclusive choice between the StartSurveillanceCamera process (id: 5) and the Final process (id: 6), as defined in Algorithm 6.

Algorithm 6: LOTOS specification for ChekPatientCondition process

```
process ChekPatientCondition [SEND, RECV] (Id:Int)
:exit :=
RECV !Id !3 !RUN ! void;
ExclusiveChoice [SEND, RECV] (Id insert (6, insert (5, {}
) ))
>> exit
endproc
```

The StartSurveillanceCamera process waits a run message from the ChekPatientCondition process before executing and finally starting the Final process. This realises sequence pattern as defined in Algorithm 7.

Algorithm 7: LOTOS StartSurveillanceCamera process

```
process StartSurveillanceCamera [SEND, RECV] (Id:Int )
:
exit :=
RECV !Id !4 !RUN ! void;
Sequence [SEND, RECV] (Id, 6)
>> exit
endproc
```

Finally, the Final process which is corresponding to the final node of sequence diagram will simply merge the three processes (id: 1), (id: 4) and (id: 5).

Algorithm 8: LOTOS Final process

```
process Final [SEND, RECV ] (Id:Int ): exit :=
SimpleMerge [SEND, RECV] (insert (5, insert (4,
insert (1, {}))), id)
>> exit
endproc
endspec
```

We have translated the scenario model into formal LOTOS specifications that is compliable into a mathematical representation. Formal verification tools such as CADP can then explore all possible execution branches on the mathematical model and prove the validity of temporal properties. We proceeded to the verification of these properties with the CADP tool EVALUATOR. This tool allowed us to perform verification and indicated that the properties were evaluated as 'true' and this proves that the specification is well compliant. As explained before, the considered scenario is sufficient enough for the verified properties to be obviously true. The development environment and case study presented in this section prove the concept and demonstrate the effectiveness of the approach proposed in this paper.

6 Conclusion

In this paper, we presented the Ubi-SO approach which facilitates the engineering of pervasive applications by integrating ubiquitous aspect that is context-awareness which is an important aspect in pervasive computing. This will offer a solution that it provides guidelines for studying and modelling a ubiquitous process. Compared to more methods, only a few approaches provide engineering methods based on a detailed development process. The main advantage of Ubi-SO is that provides ubiquitous process and helps developer to model his process by pulling and separating the ubiquitous aspects. In addition, the approach uses UML that allows an adaptation of the situations to the contextual environment. Finally, using of LOTOS performs verification and indicates that the properties were evaluated as true. This proves that the specification is well compliant, and so, the considered scenario is sufficient enough for the verified properties to be obviously true. This proves the validity of our approach. In our future work, first, we plan to extend our study by considering other scenarios on healthcare environment using all of the UML diagrams. Afterwards, we aim to consider other more complex domains such as Intelligent Transportation Systems (ITS) by validating their consistency and reliability with LOTOS.

References

Abowd, G.D. (2016) 'Beyond weiser: from ubiquitous to collective computing', *Computer*, Vol. 49, No. 1, pp.17–23.

Ahmed, B., Gherbi, A. and Kazar, O. (2017) 'Semantic-based approach to context management in ubiquitous environment', *Procedia Computer Science*, Vol. 109, pp.592–599.

Ameyed, D. (2017) *Modeling and formal specification of context and its prediction in diffuse systems: an approach based on temporal logic and the stochastic model*, MONTREAL.

Batarseh, F.A. and Gonzalez, A.J. (2018) 'Predicting failures in agile software development through data analytics', *Software Quality Journal*, Vol. 26, No. 1, pp.49–66.

Booch, G. (1995) *Object Solutions: Managing the Object-Oriented Project*, Addison Wesley Longman Publishing Co., Inc.

Brinksma, E. (1988) *Information Processing Systems – Open Systems Interconnection – LOTOS – A Formal Description Technique based on the Temporal Ordering of Observational Behaviour*, International Standard, ISO, Technical Report.

Bruneliere, H., Burger, E. and Cabot, J. et al. (2019) 'A feature-based survey of model view approaches', *Software and Systems Modeling*, Vol. 18, No. 3, pp.1931–1952.

Cipriani, N., Wieland, M. and Grossmann, M. et al. (2011) 'Tool support for the design and management of context models', *Information Systems*, Vol. 36, No. 1, pp.99–114.

Dos Santos, L.B.R., De Santiago Júnior, V.A., Povoa, L.V., Freitas, A.V. and De Castro Mario, C. (2019) 'Software inspections: comparing a formal method based with a classical reading technique', *International Journal of Computer Applications in Technology*, Vol. 59, No. 4, pp.296–317.

Dumez, C. (2010) *Approche Dirigee Par Les Modeles Pour La Specification, La Verification Formelle Et La Mise En Oeuvre De Services Web Composes*, Thèse de doctorat, Université de Technologie de Belfort-Montbeliard.

Escobar, G.J., Baker, J.M. and Kipnis, P. et al. (2017) 'Prediction of recurrent clostridium difficult infection using comprehensive electronic medical records in an integrated healthcare delivery system', *Infection Control and Hospital Epidemiology*, Vol. 38, No. 10, pp.1196–1203.

Ferscha, A. (2011) *Pervasive Adaptation: Next Generation Pervasive Computing Research Agenda*, Institute for Pervasive Computing, Johannes Kepler University Linz. Available online at: http://www.perada.eu/research-agenda/

Henricksen, K. and Indulska, J. (2006) 'Developing context-aware pervasive computing applications: models and approach', *Pervasive and Mobile Computing*, Vol. 2, No. 1, pp.37–64.

Shahzad, K., Jianqiu, Z., Zubedi, A., Xin, W., Wang, L. and Hashim, M. (2020) 'DANP-based method for determining the adoption of hospital information system', *International Journal of Computer Applications in Technology*, Vol. 62, No. 1, pp.57–70.

Vieira, V., Tedesco, P. and Salgado, A.C. (2011) 'Designing context-sensitive systems: an integrated approach', *Expert Systems with Applications*, Vol. 38, No. 2, pp.1119–1138.

Wang, X., Ong, S.K. and Nee, A.Y.C.A. (2018) 'Comprehensive survey of ubiquitous manufacturing research', *International Journal of Production Research*, Vol. 56, Nos. 1/2, pp.604–628.

Weiser, M. (1991) 'The computer for the 21st century', *Scientific American*, Vol. 265, No. 3, pp.94–104.

White, S.A. (2004) *Introduction to BPMN*, IBM Cooperation, Vol. 2, p.1.