Softw Syst Model (2018) 17:1169–1195
https://doi.org/10.1007/s10270-016-0550-0

REGULAR PAPER

A model-driven development approach for context-aware systems

Imen Jaouadi[1,2] · **Raoudha Ben Djemaa**[2] · **Hanêne Ben-Abdallah**[3]

Received: 29 April 2015 / Revised: 13 June 2016 / Accepted: 15 July 2016 / Published online: 13 October 2016

Abstract The widespread usage of various types of computer devices with different platform characteristics created a need for new methods and tools to support the development of context-aware applications capable of dynamically adapting themselves to context changes. In this paper, we present a new model-based approach that addresses the development of context-aware applications from both the theoretical and practical perspectives and that supports all development phases of context-aware systems. On the one hand, we describe how our approach is applied to dynamically capture, observe the change of the context and notify the system at runtime. On the other hand, we show how our approach is used by programmers to develop a context- aware application.

Keywords Context modeling · Application adaptation · Context-aware application development · Model-driven development

Communicated by Prof. Heinrich Hussmann.

✉ Imen Jaouadi
Jaouadi.Imen@fsegs.rnu.tn

Raoudha Ben Djemaa
Raoudha.Bendjemaa@isimsf.rnu.tn

Hanêne Ben-Abdallah
HBenAbdallah@kau.edu.sa

1 MIRACL Laboratory, FSEG, University of Sfax, Sfax, Tunisia

2 ISITCOM, University of Sousse, Sousse, Tunisia

3 King Abdulaziz University, Jeddah, Kingdom of Saudi Arabia

1 Introduction

The variety of computer devices and platforms, the circumstances of software uses, and the skills and preferences of the end users open a vast field of opportunities and challenges in software development. Indeed, today's software development methods are expected to deliver software that: (1) satisfies the various application domains' requirements (medical, transportation, air traffic, etc.), (2) self-adapts at runtime to the different needs and profiles of end users (novice, expert, disabled, minor, etc.) and (3) can be deployed on emerging new platforms (PDA, tablet, etc.). These multiple expectations form different *contexts of use*. In this paper, we define the context of use, or context for brief, as "any information relevant to the interaction of the user with the application, where both the user and the application's environment are of particular interest" [7]. This context definition means that, depending on their contexts, users accessing the same data/services through the same devices may receive different outputs; the difference is due to the users' profiles. Systems that are capable of adapting their interactions based on the context are called *context-aware* systems [9,17].

The development of context-aware systems differs from the development of traditional systems, and it still faces several challenges. These latter are inherent to how the context information is gathered, represented and exploited to adapt the system's interactions. Context information is often captured using sensors from heterogeneous sources that use different representations. Thereby, the context information may require additional semantic and structural interpretation to be significant for the context-aware software. In addition, because the context is dynamic, the software must have a means to monitor the dynamic context elements and detect their changes at runtime. Fur-

thermore, upon each detected change, the software must adjust its behavior in real time. Given the innumerable contexts, developing one single version of the software for each specific context is neither feasible nor scalable. Instead, the software development method should account for the context in order to deliver a context-aware application.

Indeed, many approaches have focused on the use of software engineering support (e.g., meta-modeling, architecture reuse, middleware) when developing context-aware applications [1,3,9,11,13,16,18–21]. Their objective is to enrich software engineering approaches for traditional software to make them capable of properly accommodating requirements inherent to different context scenarios while generating automatically different versions of the same application. Nonetheless, existing approaches have some shortages: Some of them did not rely on rich context models (e.g., [3,9]); others did not support the dynamic changes in context elements [e.g., [1,19,20]); yet others only offer a development methodology (e.g., [13,21])]. In other words, none of the existing approaches provide a comprehensive solution to context-aware application development. Such a solution addresses the development of context-aware applications from both the theoretical and practical perspective and supports all of their development phases. The herein proposed approach is one step toward the definition of such solution. More specifically, we propose in this paper a new model-based approach for the development of context-aware applications called Context-Aware Application Development Approach (CAADA). To validate this approach, firstly, we developed the framework Dynamic Observation and Notification framework for Context changes In Runtime (DONCIR), a software framework that provides for monitoring dynamically context changes and notifying the system at runtime. This framework has the merit of defining a rich context meta-model that represents dynamic and behavioral aspects of the context. Besides refining the meta-model that was initially proposed in [12], we propose in this paper a development process to support the development stages of context-aware adaptable applications using the framework DONCIR.

To sum up, our herein presented work shows contributions on three axes: conceptual, theoretical and practical.

On the conceptual axis, we detail a domain-independent meta-model for describing and monitoring the context in terms of its dynamic and behavioral aspects.

On the theoretical axis, we define the three-layered architecture of a new model-based approach for the development of context-aware applications called Context-Aware Application Development Approach (CAADA): context management layer, context change management layer and adaptation management layer.

Finally, we present two practical contributions. The first is a Java API of the framework DONCIR that is capable of capturing the context, observing it in runtime, discovering events that change it and triggering actions to adapt appropriately the running application. The second contribution is a development process that covers all the activities related to the context modeling and development of context-aware applications to use our framework DONCIR. To show the feasibility of our development process and correct functioning of our framework DONCIR, we report on the development of an application in the healthcare domain.

The remainder of this paper is organized as follows: Sect. 2 presents related works on context-aware application development. Section 3 presents the architecture of CAADA. Section 4 details the context meta-model used in DONCIR which is overviewed in Sect. 5. Section 6 details the proposed development process. Section 7 illustrates the feasibility of DONCIR through the development of a healthcare application. Finally, Sect. 7 summarizes the presented work and outlines its extensions.

2 Related work and discussions

2.1 Related work

A lot of work has been done in the area of context-aware computing in the past few years. This section presents a selection of works that focus on building context-aware applications based on models.

Seminal work was done by Dey et al. [9] in defining an architecture for building context-aware applications. It developed a context toolkit that enables rapid prototyping of context-aware applications. The architecture of the context toolkit is composed of: sensors to collect context information; widgets to encapsulate the contextual information and provide methods to access the information; interpreters to transform the context information into high-level formats that are easier to handle; and aggregators to group the context information related to a subject or situation. This architecture is simple to implement and allows separation of the acquisition process from the context representation and adaptation. However, it is not based on a conform context model for organizing the wide range of possible contexts in a structured format, nor for writing rules about contexts.

WildCAT [6] is an extendable Java framework that offers mechanisms for developing context-aware applications. It defines a dynamic data model to represent the execution context for several application domains, which separates different aspects of the execution context. In addition, WildCAT offers an programming interface to discover, interpret and

monitor the events occurring in an execution context and to record every change occurring in the context model. To support the context modeling, WildCAT defines a dynamic context model, but it does not implement any of the mentioned context domains; it just provides interfaces to realize them. Thus, WildCAT presents a programming interface, but it does not have proposed mechanisms for the deployment and the configuration to help programmers to use this framework. Furthermore, it does not provide a means to gather the data, and it rather provides methods to monitor the sensors.

In the same context, [3] proposed a framework called Java Context-Awareness Framework (JCAF) based on the Java language. JCAF is a service-oriented, event-based infrastructure suitable for the development of context-aware applications. It is based on the following component: *context clients*, *context service*, *access control, monitor* and *context actuator*. The *context clients* are the context-aware applications using the JCAF infrastructure through subscribing or requesting context information or by subscribing to an entity listener. The *context service* is responsible for handling a specific context. The access to a context service is controlled through the *access control* component, which ensures correct authentication of client requests. A *monitor* is responsible for the communication with sensors to acquire context information. A *context actuator* is designed to work together with one or more actuators to change the context. JCAF models the context through the following classes: *Entity*, *Context*, *Relationship* and *ContextItem*. It is extendable and it supports adaptation in runtime based on events. However, the context representation of JCAF does not provide the ability to define other abstractions over than simple context information. In addition, JCAF aids technology expert, but it certainly does not assist novice users; it does not offer mechanisms for the development or the configuration to help programmers to use JCAF. Furthermore, JCAF supports various sensors for monitoring locations and base classes for describing relevant entities used in context-aware applications. However, it is very hard to extend it with new computational resources to cope with an increased number of sensors.

Costa [5] proposes an architecture based on the Event–Control–Action and composed of three components, namely the *context processor*, the *controller* and the *action performer*. The *context processor* component gathers context information from the user's environment, performs context reasoning and generates context and situation events. The *controller* component observes events from context processors, monitors conditions rules and triggers actions on action performer when the condition is satisfied. To support this architecture, [5] proposes a context model based on three foundational concepts: *Entity*, *Context* and *Context Situation*. This approach does not consider reusing existing application models and context models to be developed from scratch as a new separate model. In addition, the used context meta-model does not offer important concepts to describe the dynamic aspect of the context such as *focus, temporal constraints* and *the associations types* that are defined by the work reviewed in [12]. Furthermore, this work proposes only a design methodology for structuring context-aware applications that is based on the service-oriented architectural style; in particular, it does suggest any concrete implementation.

Vieira et al. [21] propose a framework named CEManTIKA to support domain-independent context modeling and context-aware system design. CEManTIKA is composed of *context source*, *context manager* and *context consumers*. Each *context source* provides a specific context element. The *context manager* controls the activities related to context acquisition, processing, dissemination and storage. In addition, it manages the context sources used by the context-aware system. The context consumers change their behavior according to the condition related to the context. This approach is based on a generic context meta-model which models the structural and behavioral aspects of the context. However, this meta-model does not model the relationships among the entities and the histories of context information. In addition, CEManTIKA proposes only guidelines to support the design of context-aware systems; [21] does not propose any implementation or code for development.

The conceptual framework named TriPlet proposed in [13] supports the adaptation of user interfaces of interactive systems. It is structured in three core components: a *Context-Aware Meta-model* (CAM), a *Context-Aware Reference Framework (CARF)* and a *Context-aware Design Space (CADS)*. CAM defines concepts required to implement and run a context-aware application. CARF is a reference framework created to list the most relevant concepts for implementing and executing context-aware applications. It can be used either before the implementation phase as an extensive catalogue to guide developers in taking design decisions, or after the implementation phase to analyze and evaluate the concepts that were considered, which helps to identify underexplored areas for future extensions. The Context-aware Design Space (CADS) component analyzes, compares and evaluates coverage levels of the application's user interfaces through a set of well-defined criteria. CADS supports stakeholders in the phases of implementation, analysis and evaluation of adaptive and adaptable applications. Similar the aforementioned two approaches, this approach is limited to offering only a methodology for building applications, with no proposed implementation, code, nor development or configuration mechanisms to help programmers to develop applications.

2.2 Synthesis and motivation

In order to characterize and compare the works related to our proposal, we use a set of criteria. Some of these criteria are inspired from the works reviewed in the previous section, and others have been identified based on the requirements of our problem.

- *Context model*: The data model used to represent the context is examined in terms of the following criteria:
  - *Domain independence*: The context model should be as generic as possible to widen its applicability in different application domains.
  - *Expressive power*: It must be rich enough to represent all relevant aspects of the context. In particular, we focus on the following subcriteria:
    - *Dynamics*: It allows to represent the dynamic aspects of the context.
    - *Behavioral and structural information*: It should allow to model the structural and behavioral aspect of the context.
- *Architecture*: To support the building of context-aware applications, an approach should support the following elements:
  - *Context collection*: It gathers the context information from different data sources attached to the application.
  - *Context interpretation*: It transforms the context information collected from data sources into significant format easier to handle.
  - *Behavior specification*: It specifies the different application behaviors in terms of context information changes.
  - *Context control*: It observes the dynamic elements of the context and detects changes that occur in the context.
  - *Application adaptation*: It adapts the application to the constantly change in the context.
  - *Separation of context management and adaptation*: This separation of concerns provides for reusability and extendibility of the approach.
- *Validation of the architecture*: To validate the proposed architecture, an approach should provide for:
  - *Detailed development process*: The approach must propose a development process with tools to support the development stages of context-aware applications.
  - *Rapid development and configuration of applications*: The approach should offer easy-to-use mechanisms for the development and configuration of applications. These mechanisms should not require much programming and configuration efforts. Subsequently, the development and configuration time should be reduced with respect to the time required to develop an application without the approach of context-aware development.
  - *Support for implementation*: The approach must exist concretely as a usable implementation.

Table 1 synthesizes our analysis of the aforementioned approaches in terms of the above criteria.

Based on the summary provided in Table 1, we can conclude that no single approach addresses all the issues identified in our criteria. Although the examined approaches took into account, to some extent, some issues related to the creation of context-aware applications, they also present a number of limits and constraints. Indeed, most of the proposed work either did not use/propose rich context models [3,6,9], or they did not offer solutions to integrate the behavioral aspects of context modeling. In addition, those approaches supporting the dynamic, structural and behavioral aspects of the context have only proposed methodologies for designing context-aware applications [5,13,21]; the exceptions are [3,6] who implemented their framework. Moreover, we observed that no approach has been reported on the development of context-aware application that combines an integrated support on context modeling, an architecture that supports all the features of a context-aware application and a support for implementation.

As we present in the following sections, we address the above- mentioned limits through the proposition of Context-Aware Application Development Approach (CAADA), a model-driven approach for context-aware application development.

3 Overview of CAADA

As depicted in Fig. 1, the CAADA approach is based on an architecture consisting of three layers, namely *context management*, *context change management* and *adaptation management*.

The context management layer captures the contextual data of the application from the context sources based on a context model. In addition, this layer can detect and analyze context changes in the real time. To do so, we use two types of software modules: *Context Providers* and *Controllers*. The *Context Providers* collect context information from different heterogeneous sources and reformulate it to a structure understandable by the system of context-aware application. The *Controllers* monitor the context information sent by the *Context Providers*. To detect context changes

Table 1 Comparison of CAA development approaches

Criteria	Approaches					
	Context Toolkit [9]	Wildcat [6]	JCAF [3]	Costa et al. [5]	CEManTIKA [21]	TriPlet [13]
Context model						
Domain independence	+	–	+	+	+	+
Expressive power						
Dynamics	–	+	–	+	+	+
Behavioral and structural information	–	–	–	+	+	+
Architecture						
Context collection	+	+	+	+	+	–
Interpretation context	+	–	–	–	+	–
Behavior specification	+	–	–	–	+	–
Context control	–	+	+	+	+	–
Application adaptation	–	+	+	+	+	–
Separation of context management and adaptation	+	+	–	+	+	+
Validation of the architecture						
Detailed development process	–	+	–	+	+	–
Rapid development and configuration of applications	–	–	–	–	–	–
Support for implementation	–	+	+	–	–	–

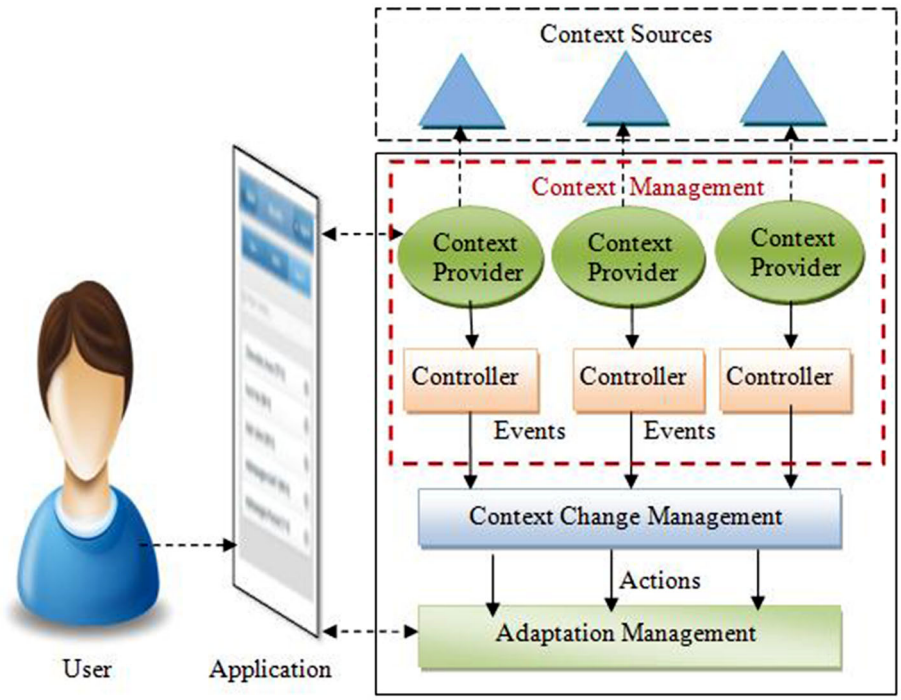


Fig. 1 Overview of CAADA approach

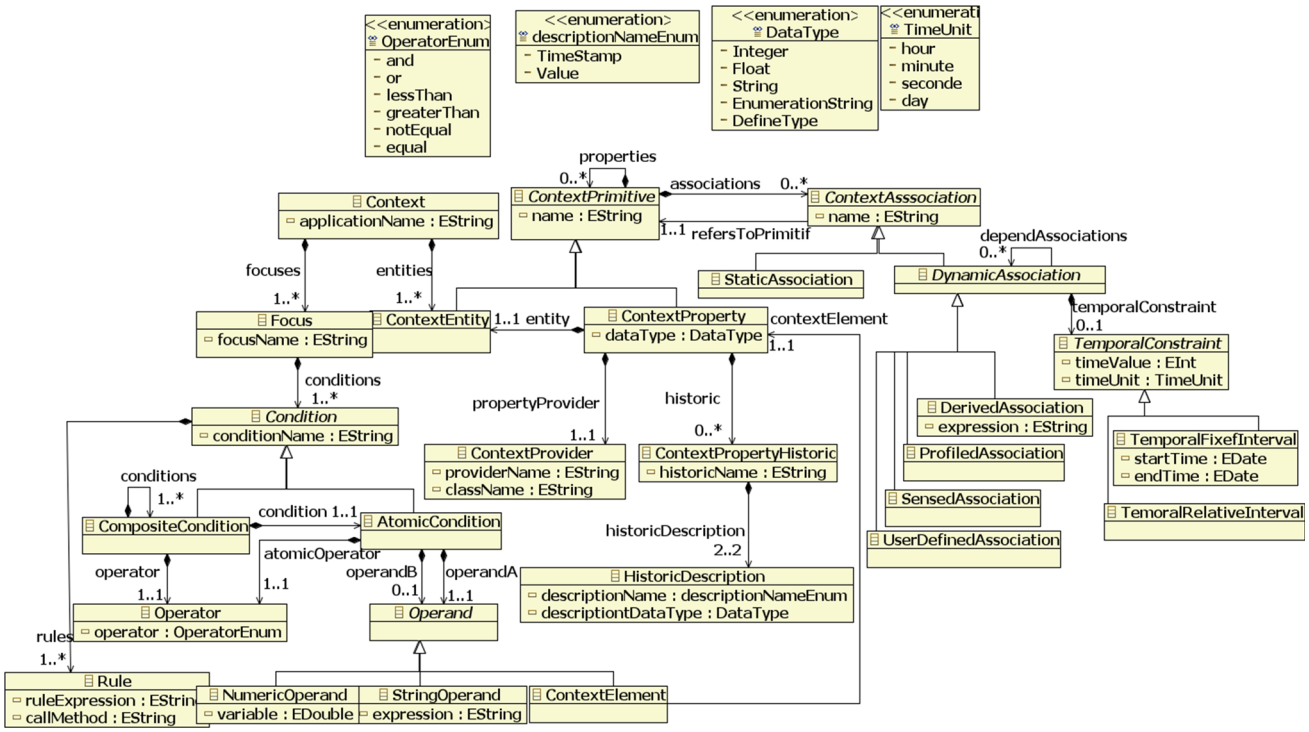


Fig. 2 Context meta-model of DONCIR

and notify the system, our approach uses the paradigm ON Event if Condition do Action (ECA): When an event occurs, the condition is evaluated and, if verified, the action is triggered. When the *context change management* identifies the context change type, it sends the necessary reactions in response to the *adaptation management Layer*. This layer, on the one hand, identifies the type of adaptation required according to the reactions defined by the *context change management* and, on the other hand, it realizes this adaptation.

The adaptation aims to change one or more aspects of the interactive system according to the context in which end users are located.

To validate this approach, we have developed a software framework called DONCIR that observes the execution context of an application and detects events that can trigger adaptation in runtime. DONCIR uses a rich generic and rich context meta-model that represents the static and dynamic aspects of the context (see Sect. 4). Also, to help and guide the programmer to develop their application as part of framework DONCIR, we define a development process with configuration and design tools (see Sects. 5, 6).

4 Context meta-model

A meta-model enables a semiautomatic generation of an application, reducing development efforts and facilitating the reuse. Our meta-model (Fig. 2) conforms to the MOF language [14] which is a standard that specifies the syntax and structure of all meta-models used. In this paper, an overview of the context meta-model is done, and for a more detailed discussion about the expressive power of the meta-model and its validation, interested readers can refer to [12].

- *Context*: The context class is the container of the elements of context model. Its aggregation association defines the containment relationships with the rest of the elements.
- *ContextPrimitive*: The *ContextPrimitive* class is an abstract class that represents the superclass of *ContextEntity* and *ContextProperty* classes. Each *ContextPrimitive* is described by a *name* and can be composed of properties. The *ContextPrimitives* are linked by associations. The *ContextEntity* class describes any physical or conceptual object, from which context information is captured, for example: doctor, room, patient and device. The *ContextProperty* class describes the properties of a ContextEntity. Each *ContextProperty* is described by a *name* and a *dataType*.
- *ContextProvider*: This class is useful to describe the provider of properties of ContextEntity. The *providerName* attribute describes the provider of context, which can be a sensor or database to search for information on the user profile. The *className* attribute describes the name of the class responsible to capture context information.

- *ContextPropertyHistoric and HistoricDescription*: Applications can make use of not just the current context, but also the past contexts to adapt their behavior for better interacting with users. Thus, we store contexts continuously, as they occur, in a database. Since all context events have a well-determined structure, it is relatively simple to automatically develop schemas for storing them into a database. The *ContextPropertyHistoric* class represents the values of ContextProperty which can take them during its lifetime. Each history of the context information is described by two attributes called *HistoricDescription*. This latter is a simple couple *(DescriptionName, descriptionDataType)*. The *DescriptionName* attribute is an enumeration formed by *TimeStamp* and *Value* attributes. The *TimeStamp* attribute indicates the time during which the value is stored in *ContextRepository*, whereas the value of *ContextPropertyHistoric* is indicated by the "*value*" attribute. The *data type* of each attribute is represented by the attribute *descriptionDataType*.
- *ContextAssociation*: It is used to describe the relationship between two entities or two properties and the relationship that link ContextEntity to its properties. These types of relationships are represented by the *refersToPrimitif* association. Each association is described by a *name*. Associations are classified into two groups: The *StaticAssociation* represents relationships which remain fixed throughout the life of *ContextEntity*, such as date of birth. The *DynamicAssociation* represents all associations that are not static. They are classified into four groups: The *SensedAsssoiation* represents information obtained through sensor such as doctor location. The *DerivedAssociation* describes the values derived from other associations. For example: The patient's age is derived from the patient's birth date and the current date. The room number that the doctor is going to visit is deducted from the location of the doctor. The *ProfiledAssociation* represents the information extracted from the user profile such as the patient's date of birth. While the *UserDefined* represents the values provided by the user through a dialogue interface such as mood or motivation of a person which allows the system to make decisions. A *DynamicAssociation* can depend on other associations. This is indicated in the diagram by the reflexive relationship *dependAssociations*. In addition, a dynamic association can be defined by temporal constraints. A *TemporalConstraint* is described by a value and unit *(TimeUnit)* used to represent this value. There are two groups of *TemporalConstraint*. The *TemporalRelativeInterval* class is used to define a time interval based on the current time. The *TemporalFixedInterval* class is used to explicitly define the interval time. The *startTime* attribute describes start time and *endTime* describes the end time.
- *Focus*: A context-aware application is composed of several Focuses. In Zimmermann et al. [21,23], the authors show that the context cannot be considered in an isolated manner, but always connected to a focus. A focus is determined by the *focusName*. For example, a user can interact with the application to look for trains depart list. If a focus is active, it triggers one or more conditions.
- *Conditions*: It is the superclass of the *AtomicCondition* and the *CompositeCondition* class. The *AtomicCondition* class represents the atomic Conditions, while *CompositeCondition* class represents the composed conditions. Each condition is defined by a name and a value calculable. Each condition represented by the *AtomicCondition* class is defined by two operands called *operandA* and *operandB* and one of the following operators (and, or,>, <, =, <>). An operand is represented by the abstract class *Operand*; it can be a numeric variable *(NumericOperand),* or a string *(StringOperand)* or a context element *(ContextElement).* A composed condition consists of several operators *(compositeOperator)* and one or more atomic condition represented by *atomicCondition* relationship. Example of composed condition is: ((Noise-Level='high') and (doctor-Experience>2) and (Time='Morning')).
- *Rule*: A rule specifies a procedure to be executed when the conditions of active focus are met. It activates the execution of an adaptation. A rule is described by the *RuleExpression* attribute that describes the expression of the rule. The *callMethod* attribute describes the function to be performed to realize the adaptation enabled by the rule.

Besides its genericity (i.e., all modeled concepts are domain independent) and conformity to MOF [14], our meta-model is richer than the proposed meta-models: Firstly, it represents the context structural aspect with these concepts: *ContextEntity*, *ContextProperty* and *ContextAssociation*. Secondly, it provides for representing the context dynamic aspect with *DynamicAsssociation*, *TemporalConstraint* and *ContextPropertyHistoric* concepts. Thirdly, it provides the specification of the behavioral aspect of the context with *Focus, Condition* and *Rule* concepts.

5 Overview of DONCIR's implementation

To validate our approach CAADA, we have implemented the framework DONCIR. This framework is based on the paradigm Event Condition Action. So, to describe the functioning process of DONCIR, it is necessary first to define types of event that can be trigged in the context-aware application.

5.1 Event types in DONCIR

Each change occurring in context element is represented by an event named *ContextEvent*. Each event is identified by *oldValue*, *newValue* and an Event source (an object that indicates the origin of the event). In addition, the dynamic change of context in runtime can change the properties and the conditions values. Our framework should consider these changes. It provides a JAVA interface, named *ContextChangedListener*, as the superclass of *ContextValueListener* and *ConditionListener* interfaces. The change in ContextProperty value is triggered by the *propertyValueChanged* method of the *ContextValueListener* interface. The *ConditionListener* interface describes the methods used to manipulate events triggered when changing a condition value. The *conditionoccurred* method is invoked if a condition is changed value to true. The *conditionRemoved* method is called if a condition changed value to false. To deal with these events in our implementation, we used the *EventListener* class. It is a java interface that inherits from the superclass *java.lang.Object*. In fact, a class which wants to listen to contexts elements change will be able to register with the model as *EventListener*. In addition, the class must have methods to add and to remove listener, and methods to signal a change usually have the prefix "fire" in the name. Listening 1 shows the *ContextProperty* class that is registered as *EventListener* and indicates the change in property value by the *fireContextValueChanged()* method.

Listening 1 : Extract of ContextProperty class of framework DONCIR

```
public class ContextProperty extends ContextPrimitif
{protected ContextEntity  entity;
 protected DataType dataType;
      protected ContextProvider provider;
 protectedObject value;
 protected boolean contextValueListener=false;
 public final EventListenerList listeners=new EventListenerList();

 public void addContextValueListener(ContextValueListener listener)
 {listeners.add(ContextValueListener.class, listener);
  contextValueListener=true;
 }
 public void removeContextValueListener(ContextValueListener listener)
 {listeners.remove(ContextValueListener.class, listener);
  contextValueListener=false;
  }
 public void fireContextValueChanged(Object oldValue, Object newValue)
 {ContextEvent event=null;
  if(! Equals (oldValue ,newValue))
  {getProvider().getClassName().setValueChanged(true);
   for( ContextValueListener listener: getContextValueListner())
   { event=new ContextEvent(oldValue, newValue, this);
    listener.propertyValueChanged(event);
   }}
}
```

5.2 Functioning process of DONCIR

Figure 3 illustrates the operating principle of DONCIR.

As soon as the user opens the application and selects its target (step 1) (for example, a doctor wants to review radiology and pharmacy result), the context Generator manipulates the XML File of the context model and generates at the

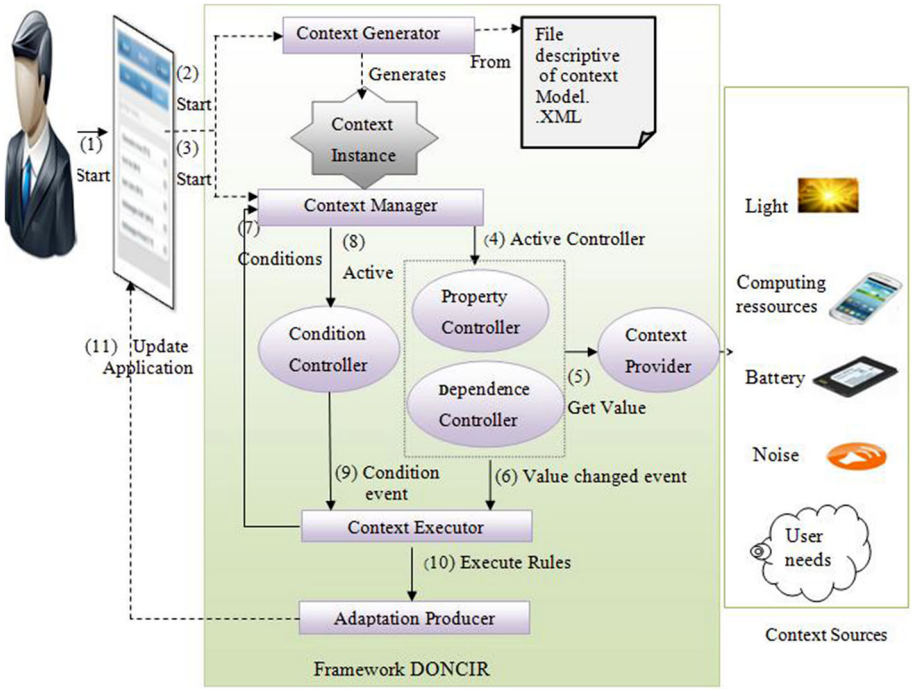


Fig. 3 Overview of DONCIR

end a context instance of the application launched by the user (step 2). When the context generator completes its task, the *context Manager* searches the context elements that may change during the execution of the application based on the instance created by the *ContextGenerator* (step 3), and it associates it with a specific controller (step 4). From this

XQuery is a declarative programming language that can be used to extract information from an XML document in much the same way as SQL extracts information from a relational database. We used Saxon as an Xquery processor because it is free.

Listening 2 describes the *extractProvider* function.

Listening 2 : extractProvider method

```
public ContextProvider extractProvider(ContextMeta-modelPackage.ContextEntity
ce,String chemin, String namep)
{XQuery xq=new XQuery(ContextModelFileName);
ContextMeta-modelPackage.ContextProvider p=null;
String query="for $a in (doc
('contextModel.xml')/ContextAwareApplication/ContextEntity[@name='"+
ce.getName()+"']/"+chemin+"/ContextProperty[@name='"+namep+"']/ContextProvider)
return (text{$a/@providerName}, text{' '},text{$a/@className},text{'&#xA;'})";
try
{//write return of executequery inFile
BufferedReader bp=new BufferedReader(new FileReader(xq.executeXquery(query)));
String sp="";
 //read file using StringTokenizer
if(!(sp=bp.readLine()).equals(" ")){
StringTokenizer ssp=new StringTokenizer(sp," ");
 //read nameProvider
String nameProvider=ssp.nextToken();
//read nameClass of provider
String nameClass=ssp.nextToken();
// create new instance of contextProvider
try {
p=new ContextMeta-modelPackage.ContextProvider(nameProvider,
(ContextProviderPackage.ClassProvider) Class.forName(nameClass).newInstance());
} …….
```

moment, the Controllers (*propertyController*, *dependence controller*) monitor the dynamic context elements. They examine the property value of *ContextProvider* specific to each *ContextProperty* (step 5) and compare it with the previous situation. When a context change occurs, the controller triggers a *ContextValueChanged* event to the *ContextExecutor* (step 6). This latter asks the *ContextManager* to activate all *conditionController* (step 7) possible to change their values in this situation. If a condition has changed a value, the *ConditionController* sends a *conditionOccurred* or a *conditionRemoved* event to the *ContextExecutor* (step 9) to take the necessary actions. This latter sends the equivalent rules to the current context to the *AdaptationProducer* (step 10) to adapt the application and to make it consistent with the new context.

In the following sections, we detail each element of our approach and we describe how it functions.

5.2.1 The context generator

This component generates *automatically* the context model description, as an instance of context containing the functional elements and the descriptive elements of the context model based on a XML file. The structure of this file corresponds to the structure of the context model (see Sect. 4). This generator is based on Xquery language [22].

It (Listening 2) creates and returns a *contextProvider* instance from XML file providing as parameters names of *ContextEntity* and *contextProperty* of this *contextProvider* and its path in the XML file. A *contextProvider* instance is described by *providerName* and *className*. To describe the *className* of a provider, we have used the principle of reflection in Java; it allows loading a class, creating an instance and accessing its methods without knowing the class in advance. The static method *Class forName (String name)* allows to load a class whose full name is specified. Finally, the *newInstance ()* method creates a new instance of the class by calling the constructor with no parameters.

Each context instance created by the *Context Generator* will be used by the *Context Manager*.

5.2.2 The context manager

The *Context Manager* is a process whose purpose is to know the dynamic context elements as observed by the controllers. The operation principle of the context meta-model is based on the context meta-model structure as explained by the graph shown in Fig. 4. Each entity is related to its properties and other entities through dynamic associations. In addition, each property may be related to other properties. The dynamic associations are several types, and a property can be an operand of a condition.

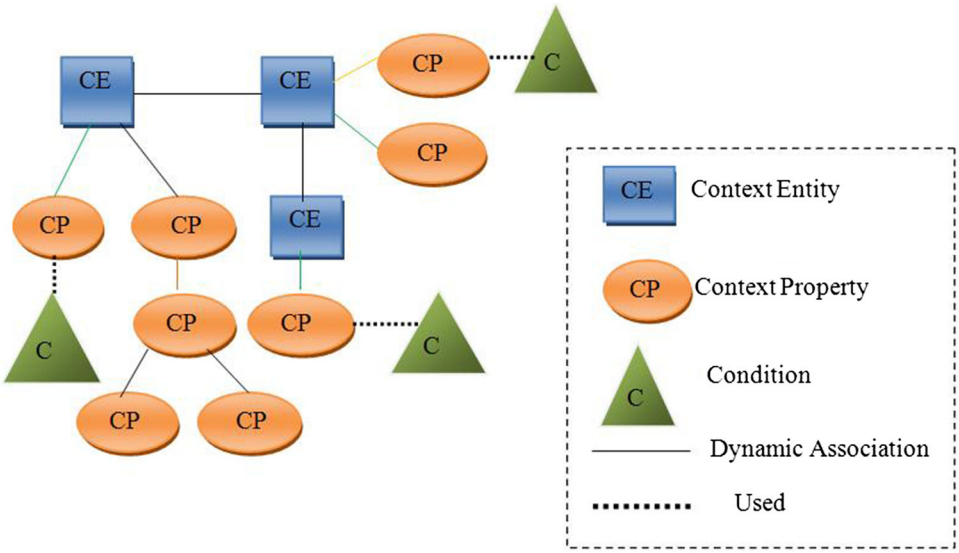


Fig. 4 A graph that summarizes the context meta-model

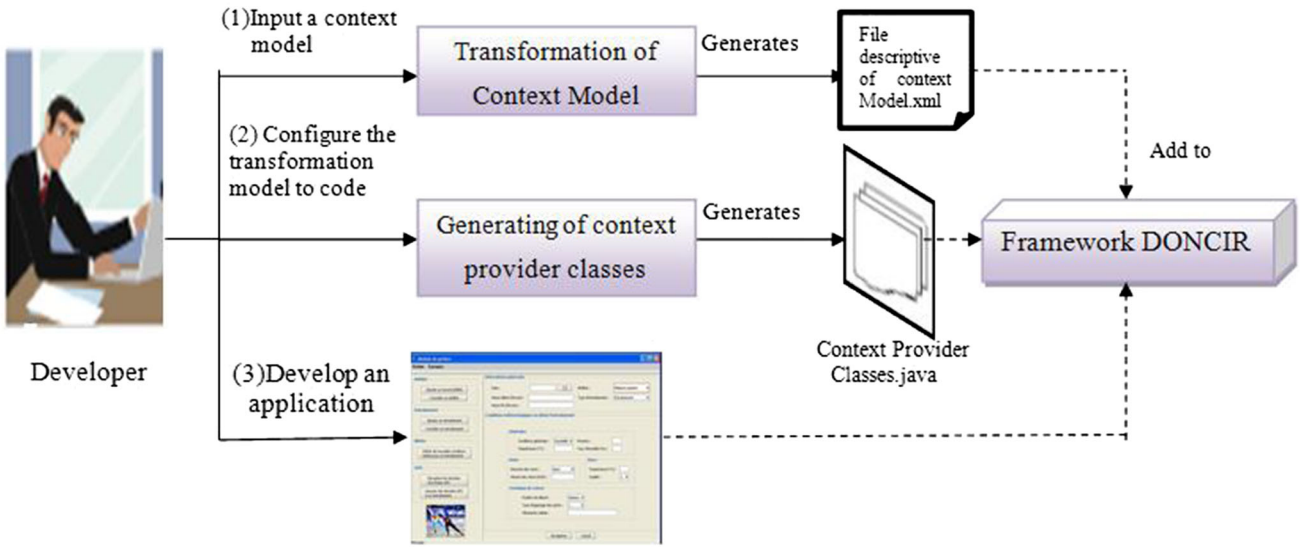


Fig. 5 The development process stages

To know the dynamic context elements, the *ContextManager* must follow these steps. Firstly, it gets the context entities list of the application in runtime. Secondly, for each entity, it searches all dynamic associations that come out. Thirdly, for each association, he looks the context element type of the association end. If the context element type is a *ContextProperty*, it adds *ContextValueListener* to the current property (CP) to listen for events when the value of property changes. Then, if the dynamic association (DA) output of CP was dependent on other associations, in this case, the change of the property value depends on changing the association's properties which depend on this change. Hence, it controls the change of these properties value by activating the dependence controller. Otherwise, the Manager adds a new *propertyController* on the property according to the time constraint used to represent the temporal aspect of dynamic associations. Finally, if *ContextProperty* is composed of other properties, the manager must repeat the previous steps for each subproperty.

The next section describes the operations performed by the controllers when they are launched in work by the *ContextManager*.

5.2.3 Controllers

Some context elements change rapidly. Consequently, this type of context elements must be continuously monitored so that it can detect changes and trigger adaptations. To do so, we use the modules called *ContextController*. A controller searches for and selects context sources and it observes context changes sensed by the selected context sources. Each *ContextController* must be able to assess the context to sig-

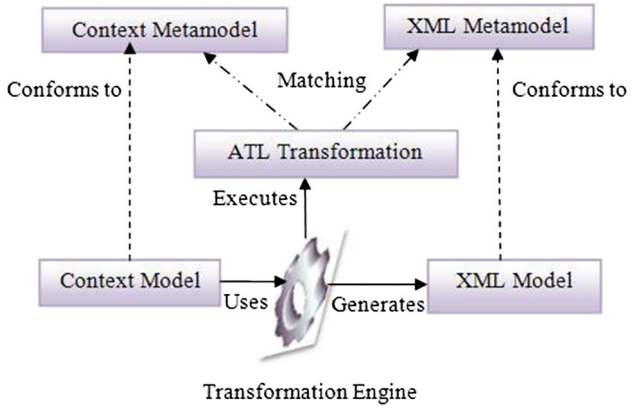


Fig. 6 Mechanism of transformation based on MDA approach

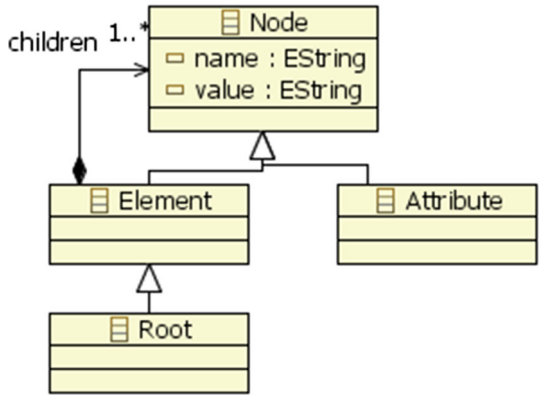


Fig. 7 XML-meta-model

nal events to *contextExecutor*. As such, we distinguish three types of controllers:

- *propertyController*: A process that controls *Context Property* value changes. When the *propertyController* is activated by the *ContextManager*, the *ContextProvider* provides each time the new context property value to *propertyControlle*r. It is the *propertyController* which detects changes by comparing the current value with the previous value and it reports the *ContextExecutor* of a trigger of a new *propertyValueChanged* event type.
- *DependenceController*: A process that controls a property $p1$ which in turn depends on the property $p2$. Each time the *dependenceController* of $p1$ detects a change in $p2$, it gets the new value of $p1$ from its context provider. For example, a roomNumber depends on the user location. Each time when the value of the user's location is changed, the *DependeceController* created for the roomNumber property requests the new value of the property roomNumber from the RoomNumberProvider class.
- *ConditionController*: A process in charge of controlling a specific condition. Each time where it will be activated by the *ContextManager*, it evaluates the condition value by comparing it with the old value. If the value of the condition is changed, a new *ContextEvent* of *ConditionOccurred* or *conditionRemoved* type will be created.

5.2.4 ContextExectuor

The *ContextExecutor* allows the system to react when a context event is generated. In our work, we have introduced two types of events: (1) *Events generated* when a property value is changed (valueChangedEvent), and (2) Events generated if a condition changes value.

A change of *ContextProperty* value is signaled by *PropertyController* sending a Value Changed event. This event will trigger the execution of the *propertyvaluechanged* function by the *Context Executor*. The change of a property value can generate the change of condition value. In the light of this idea, the *Context Executor* should search all conditions of the focus activated by the user and one of their operands is formed by *Context Property* triggering this event. Then it asks the manager to activate controllers of all these conditions. If any of these conditions become true or false, a *conditionRemoved* or *conditionOccurred* event type is triggered. Every Time the *Context Executor* receives the *conditionOccurred* event, it retrieves rules that are triggered by the conditions evaluated to be true and sends these rules to *Adaptation Producer*.

5.2.5 Adaptation producer

The *Adaptation Producer* is responsible for executing rules to take place when a condition change occurs. Each rule is defined by the *callMethod* attribute which describes the name of the function to execute to adapt the system. The *Adaptation Producer* calls the function whose name equals to *callMethod* and executes its code. All the adaptation functions are written by the programmer during the development and are registered in the *AdaptationFunctions* class.

In this paper, we will execute the code of the functions developed by the programmer and we will not be interested in the way in which the adaptation will be done because our framework is generic and can be used by all application types and the implementation is different depending on the type of the application, e.g., interactive applications and web services.

6 The development process in the context of DONCIR

In this section, we will present how developers can quickly build a context-aware application using the framework DONCIR by following our development process. This process meets the requirements defined in the related work: detailed development process and rapid development of applications. Indeed, all steps in this process are detailed and guided by easy-to-use tools. Figure 5 illustrates the phases that a programming engineer must follow when developing a context-aware application using our framework DONCIR.

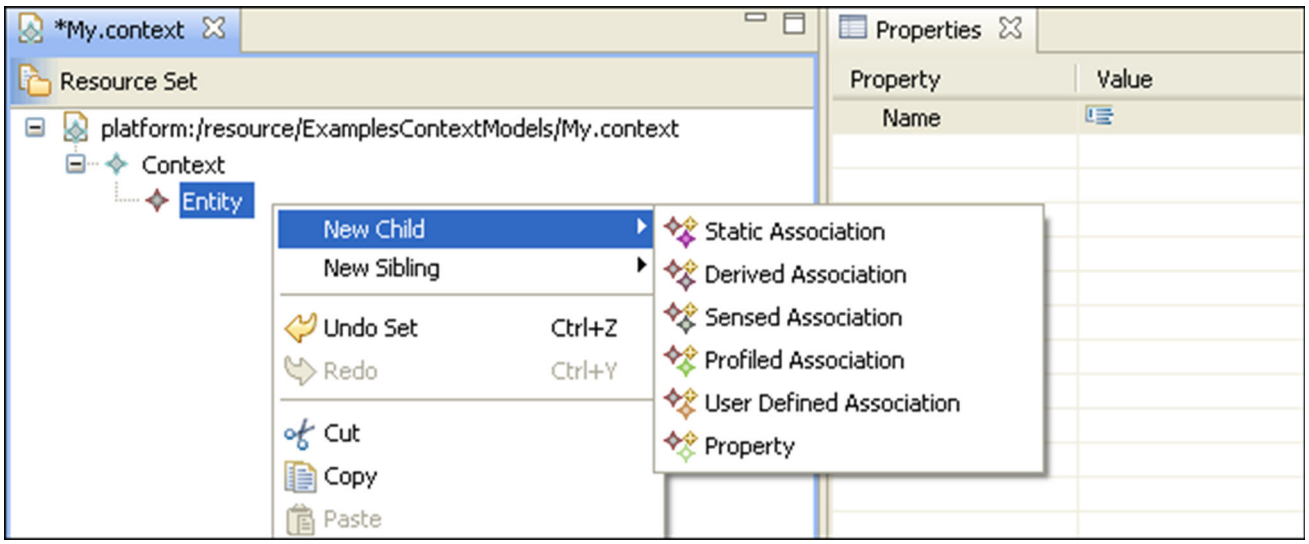
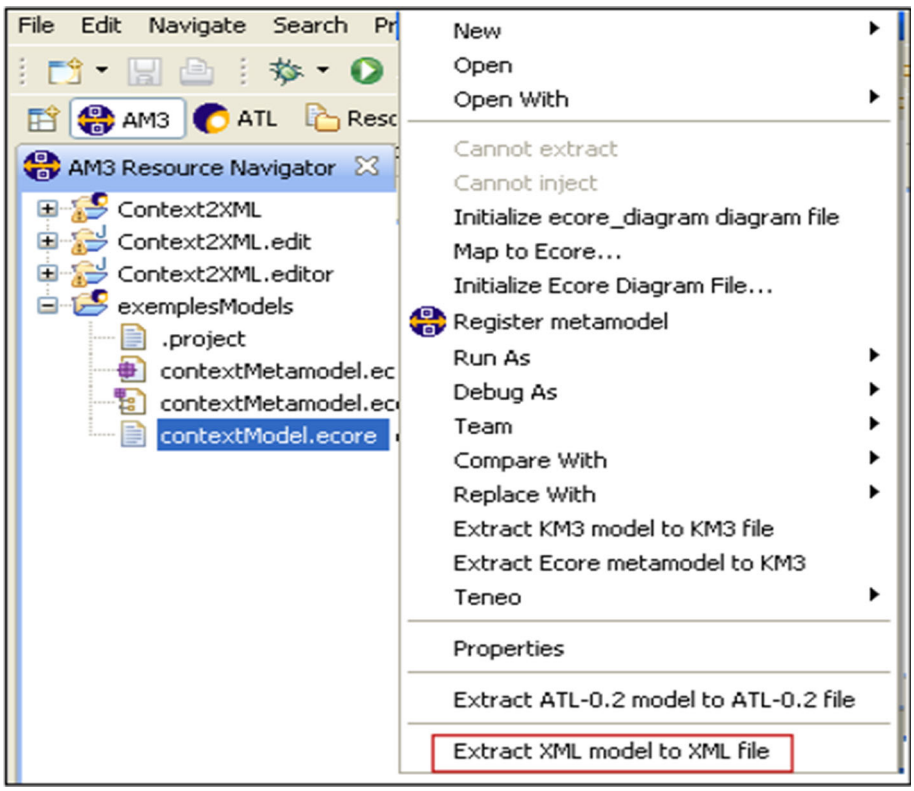


Fig. 8 A construction example of context model edited with EMF editor of eclipse

Fig. 9 The extract functionality of AM3 perspective



6.1 The context model transformation process

In our current implementation, developers of context-aware applications need to write a configuration file for describing context elements and the behavior of application. The file would be conforming to the context meta-model of DONCIR. Hence, we must provide a mechanism to automatically generate this file. This mechanism must simplify the developer's tasks and assumes the validity of the file. The first step in our approach (step1) shows the transformation process of context model to XML file. The methodology used in this step is supported by the model-driven architecture MDA [15]. This approach consists of using a set of models that can be interpreted or processed by a machine. In order to a machine can interpret or transform a model, it is necessary that the model is defined in language that is understood by it. The language used to describe the models is called meta-model. The definition meta-models of a system models are very different. To manage this diversity, the MDA recommends using a common language called meta-meta-model to describe all the meta-models involved in the description of a system model [4]. Figure 6 describes the transformation mechanism of context model to XML model.

The execution engine takes as input: the context model, the context meta-model, the XML meta-model, transformations rules to map the source meta-model to the target meta-model. It produces in output an XML model. The context meta-model is already described in Sect. 4. In the following section, we will detail the different inputs of the transformation engine.

6.1.1 XML meta-model

Figure 7 shows the XML meta-model. Each node (Node class) has a name (name attribute) and a value (value attribute)). A node can be an attribute (Attribute class), or any element (Element class) or root (Root class). An element has children (reference children) which are the nodes.

In theory, having on hand the source and target meta-models, we can write directly at this stage a transformation as an ATL code.

6.1.2 Transformation rules

The definition of transformation rules describes the correspondence between the source model elements and those of the target model. To implement our transformation rules, we have chosen the ATL language [2] because it is simple as well as flexible, and it allows hybrid rule expressions (declarative and imperative rules). Root rule is the main rule of our ATL code (Listening 3).

Listening 3: Root rule

```
1  rule Root
2  {
3  from
4   p:ContextMetamodel!Context
5   to
6   root:XML!Root(
7    name<-'Context'   )
8   do
9    {
10     for(e in p.entities)
11      { thisModule.Entities (e,root);}
12
13     for(e in p.focuses)
14     { thisModule.Focus (e,root) ;}
15   }
          16   }
```

Listening 4 : rule Entities

```
1  rule Entities(e:ContextMetamodel!ContextEntity, host:XML!Element)
2  {
3  to
4  element:XML!Element(
5   name <-'ContextEntity',
6   children <- Sequence{})
7  do
8   {
9   thisModule.Attribute('name',e.name,element);
10
11   if (e.properties->notEmpty())
12    { for(i in e.properties)
13        {  thisModule.Properties(i,element);}
14     }
15    if (e.associations->notEmpty())
16     { for(i in e.associations)
17      {thisModule.getAssociations(i,element);}
18      }
19       host.children <- thisModule.add(host.children , element);
20          }
21   }}
```

This rule aims to create the root element in the XML model (line 6) from the Context element (line 4). The key word *from* (line 3) indicates the type of the element that will be used as input for the implementation of this rule. This rule will be executed once for each element of type *Context* of source model. In the imperative block of this rule (do { } in line 8), we explicitly call the *Entities rule* (line 12), which is invoked by each "*ContextEntity*" or "*Context*" element of context model, and *Focus rule* (line 15), which is invoked each time the context element is composed of elements of Focus type (ligne14). Listening 4 presents the ATL syntax to deduct the XML element 'ContextEntity'.

Listening 5 : The Attribute rule

```
1  rule Attribute(nameAttribute:String, valueAttribute: String, element :XML!Element)
2  {
3          to
4          a: XML!Attribute(
5      name <- nameAttribute,
6      value <- valueAttribute)
7     do
8          {
9          element.children <- thisModule.add(element.children,a);
10         }
11   }
```

The Attribute rule (Listening 5) creates an attribute whose name and value are given in parameter for this rule (line 1). This rule uses a helper add (line9) for adding the attribute created at the children list end (children) of the input parameter XML element (line 1). The syntax of the *helper add* is defined by the code ATL in Listening 6.

Listening 6 : The helper Add

```
1  helper def: add(list: Sequence(XML!Element),
2   element:XML!Element): Sequence(XML!Element)=
3     if  (element.oclIsUndefined()) then list
4                 else list -> append(element)
5      endif ;
```

Helper Add is a method that has the role to add an element (line 2) entered into parameter at the end of a list (list line 1), using the operator ATL append (line 9).

In order to execute these rules by the transformation engine and generate a file, we also need a context model.

6.1.3 Context model

In this phase, designers develop a single model which addresses the possible contexts of CAA, the focuses to identify all the objects that have to be manipulated to perform including rules and conditions. The model must be conforming to the meta-model (Fig. 2). Therefore, the designer needs to ensure that the context model defined is precise and coherent. The eclipse allows using the Eclipse Modeling Framework plug-in [10] to easily create a meta-model conforming to the meta-model by instantiating the root element and then creating the elements it contain. The modeling editor restricts the designer from defining an invalid model and supports the validation of the context model. Figure 8 shows a construction example of context model conforming to meta-model edited with EMF editor of eclipse.

Figure 8 shows that the construction of a conform and valid context model is very easy and that it does not require much effort by the developer thanks to our development process. In contrast, as cited in related work, none of the existing approaches proposed a tool to quickly design context model while ensuring its validity and its compliance with meta-model.

At this stage, we have in hand: a source and a target meta-models, rules of transformation and an example of context model. So we can execute the ATL transformation. This transformation provides a target model in the XMI form [14], which conforms to the target met-model.

The ATL tool is supplied with a plug-in of global business models. This module, known by the acronym AM3 (ATLAS MegaModel Management), offers among others the functionality to extract a model to an XML file. Indeed, the ATL transformation cannot directly generate an XML file, but the extractor is thus used to directly transform an XML model into an XML file. Figure 9 shows the Extract functionality of AM3.

Similarly to the previous step, as shown in Fig. 9, the generation of an XML file from the XML model is very easy and takes only a few seconds to be done by the developer. This generated XML file will be used as input to the second step (step 2 in Fig. 5) of our process to develop a context-aware application.

6.2 Generating provider Java classes

Step2 consists of creating Java classes that inherit the *ContextProvider* class of our framework DONCIR based on the XML file generated. These classes can capture context information of each context model property.

Eclipse Acceleo is a code generation tool with Eclipse. It enables the design of code generation modules from one or more models. In our case, we generate Java classes based on the XML file generated in previous step and a little program written with markup language provided by Acceleo (Listening 7).

Listening 7: program to create Java classes from XML file

```
[module generate('http://xmlmetamodel/1.0')/]
[template public generate(element: Element)]
[comment @main /]
[for (e : element | element.children)]
  [for (a: Attribut | e.children)]
      [if (a.name='className')]
       [file (a.value.concat('.java'), false)]
              public class [a.value/]     implements ClassProvider
              {
               public static boolean valueChanged=false;
                public boolean getValueChanged()
                 {
                        return valueChanged;
                 }
                public void setValueChanged(boolean  valueChanged)
                 {
                     DoctorNameProvider.valueChanged  = valueChanged;
                 }

                @Override
                public Object getValue()
                 {
                   [comment code will be complete by developer /]
                 }
               [/file]
             [/if]
   [/for]
 [/for]
[/template]
```

In this step, the developer only needs to configure the execution of the template by selecting the source model (contextModel.xml) and the placement where the generated Java classes will be stored. This step is very easy and quick to realize and does not require any effort by the developer.

6.3 The development of an application

In this step, the programmer must develop their application and define the getValue() methods in all classes of *ClassProvider* type. In each method, the developer must specify how to capture the value of each context property from the context source.

7 Experimental study of DONCIR

To demonstrate the feasibility of DONCIR, we have considered an application in the healthcare domain as a scenario of use. For this application, we provide a high-level description of the application intended to implement the scenario and the development process following the framework DONCIR. The development process includes the context modeling activities.

7.1 Overview of the case study

In a hospital, every doctor must have a mobile that allows him to enter and to receive important information of the patient in a automated manner. Doctors work in dynamic environment. The context depends on the location of the doctor (e.g., patient room, RAI room, operating room), the time (morning and evening), the ambient conditions of doctor location (light level: bright or dark, the noise level (low or high) and the device used by the doctor. A doctor can review laboratory and pharmacy results and submit new laboratory and pharmacy orders.

In a hospital, there are two groups of doctors: The expert doctors have more experience with mobile technology and they have high tolerance toward using new applications, while the novice doctors have less experience with mobile technology. Thus, they have a difficulty reading small fonts and hearing sound low. Consequently, in normal conditions, it wants to receive information via headset and set the ring tone volume of device to high. An expert doctor has neither difficulty in reading nor hearing problem. So, in normal conditions, they want to receive information via text. We identify a number of adaptation scenarios to validate our approach. Some of these scenarios represent a particular adaptation of the application based on change in context of execution.

7.2 Application of our development process

The first step of our development process is to develop a context model of our case study and transform it into an XML file.

7.2.1 Transformation of context model to an XML file

In this step, we begin by describing the contextual elements of the case study by identifying its entities and its proprieties. Afterward, we analyze the type of each association that links two context elements to describe the acquisition mode of context element. Figure 10 represents the different entities, proprieties and associations types of our case study.

In the next step, we identify a number of adaptation scenarios to validate our approach. Some of these scenarios represent a particular adaptation of the application based on change in context of execution. Table 2 shows examples of conditions and adaptation rules to trigger for the focus "Review laboratory and pharmacy result." For example, if it is night time, and the noise level is low in a patient room, a doctor (expert or novice) who is in this room must always

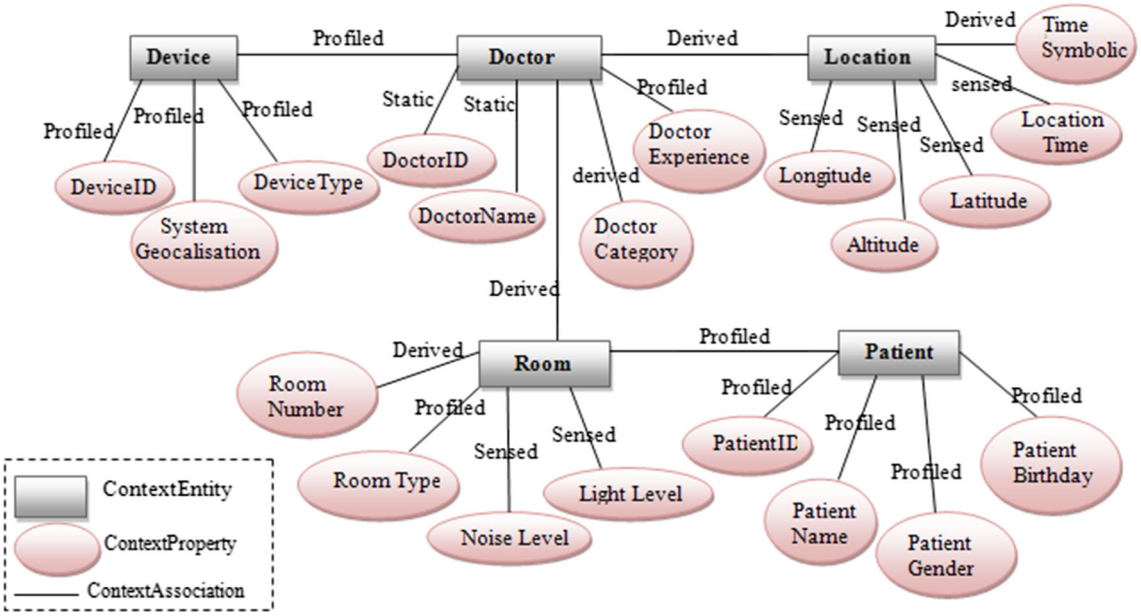


Fig. 10 Graph illustrates the context model of our case study

Table 2 Examples of conditions and rules adaptations of running example

Adaptation scenario no	Condition name	Conditions	Rules
Scenario number 1	Cond1	(Doctor category = 'novice') and (Doctor location = 'patient's room') and (Light level = 'bright') and (Noise level = 'low')	Display font size = Large Display brightness = Normal Ring tone volume = high Receive information via headset
Scenario number 2	Cond2	(Doctor category = 'novice') and (Doctor location = 'patient's room') and (Light level = 'bright') and (Noise level = 'high')	Display font size = Large Display brightness = Normal Ring tone volume = vibration Receive information via text
Scenario number 3	Cond3	(Doctor category = 'expert') and (Doctor location = 'patient's room') and (Light level = 'bright') and (Noise level = 'low')	Display font size = Normal Display brightness = Normal Ring tone volume = medium Receive information via text
Scenario number 4	Cond4	(Doctor category = 'expert')and (Doctor location = "patient's room') and (Light level = 'dark') and (Noise level = 'low')	Display font size = Large Display brightness = high Ring tone volume = silent Receive information via text
Scenario number 5	Cond5	(Doctor category = 'novice') and (Doctor location = 'patient's room') and (Light level = 'dark') and (Noise level = 'low')	Display font size = Large Display brightness = high Ring tone volume = silent Receive information via headset
Scenario number 6		(Doctor category = 'novice') and (Doctor location = 'doctors room') and (Light level = 'dark') and (Noise level = 'low')	Display font size = Large Display brightness = high Ring tone volume = high Receive information via headset

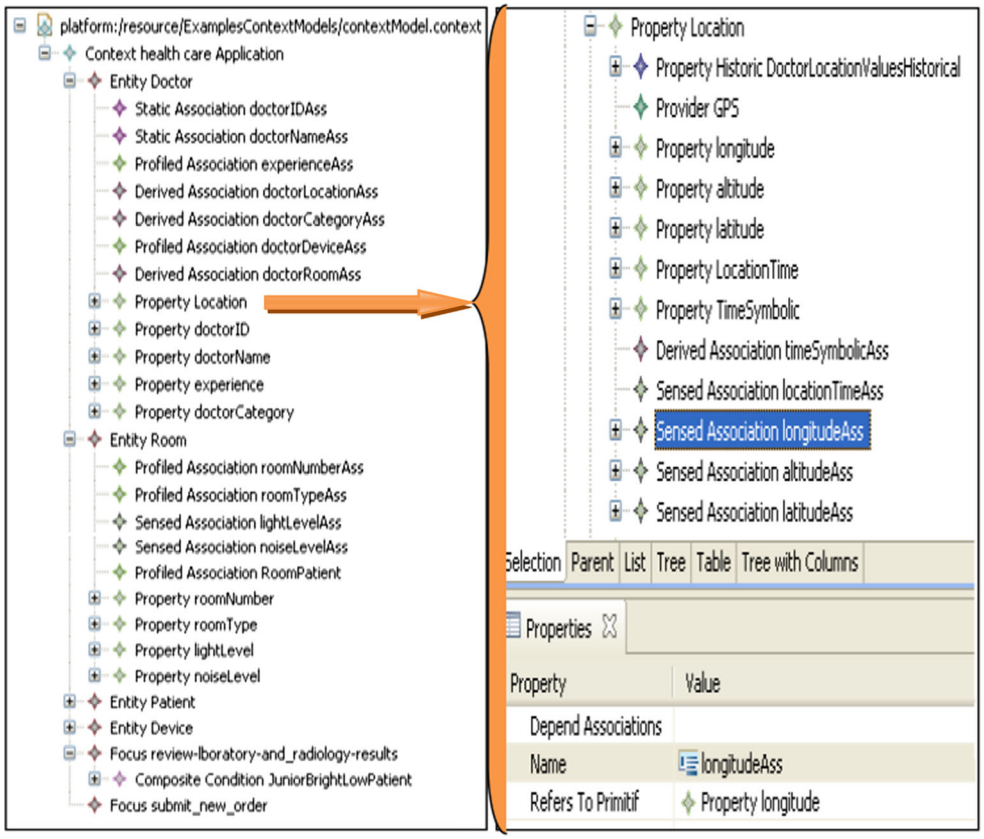


Fig. 11 Context model of study case edited with EMF

have silent ring tone volume and must receive information via text to not disturb the patient (scenarios 4 and 5).

After specifying the different context elements, we can easily design our context model with the EMF eclipse Editor. Figure 11 shows the context model of our study case.

In fact, the goal of this step is to build a conform context model that complies with the meta-model of DONCIR by following its development process. The reliability of this model is ensured by the EMF tool of eclipse with which we have built our model. Moreover, this step is rapidly guided by our building tool and it does not require much effort from the programmer.

After the construction of the context model with the editor EMF, we must transform it into an XML file. For that, we must execute the rules of transformations defined in Sect. 6.1.2. Figure 12 shows the configuration that we performed to transform the context model of case study in an XML file.

This transformation (Fig. 12) provides a target model in XMI format that conforms to the XML meta-model. Figure 13 shows an extract of the generated file during the execution of this transformation.

We then transform the generated XMI file into an XML file using the extractor of AM3 perspective of Eclipse (see Fig. 14).

7.2.2 Generating provider java classes

In this stage, we need only configure the execution of the template by choosing the source model (contextModel.xml) and the location where the java classes generated will be saved. Figure 15 shows the configuration performed by the developer of our case study and the executing result of our configuration.

7.2.3 Development of the case study application

In this step, we only develop a prototype of the case study. In our work, we construct a simple graphical application that replaces the real application and we use it to test the good functioning of the main components of DONCIR namely *ContextGenerator*, *ContextManager, Controllers*, *ContextExecutor* and *AdaptationProducer*. We will not go into the technical details of the implementation of the necessary sensors for our case study (noise sensor, position sensor, light sensor). The main objective of our proposition is to detect the change of the context in runtime and not the development of sensors. In this application, we will manually enter the information about the doctor's location, the noise level and the bright level. Figure 16 shows the main interface of our prototype. In this prototype, the launch of the application by the

user is replaced by the " run Genarator " button. The launch of the focus chosen by the user is replaced by the " start " button.

Then, the programmer must define the *getValue ()* methods in all provider java class. In each method, the developer must specify how to capture the value of each context property from the context provider. For example, Listening 8 presents the definition of the getValue () method of the *PatientNameProvider* class. This method is used to provide the name of the patient visited by the doctor. This information depends on the *roomNumber* where the doctor is.

Listening 8: getValue() method of the PatientNameProvider class.

```
public Object getValue()
{
  ApplicationExample.ConnexionBase b=new ApplicationExample.ConnexionBase();
  String patientName="";
  int roomNumber=Integer.parseInt(""+
                new ContextProviderPackage.RoomNumberProvider().getValue());

  String req= "select * from Patient where roomNumber="+roomNumber+"";
  ResultSet  rs=b.selection(req)     ;
  rs=b.selection(req);
  try
  {
  if(rs.next()) patientName=rs.getString(2);
  } catch (SQLException e)
  {
   e.printStackTrace();
  }
  return patientName;
}
```

We have presented in this section how a developer can quickly build the application of our case study in our framework DONCIR by following our development process using easy configuration and design mechanisms. In the next section, we describe the feasibility of DONCIR by testing it on the prototype developer for the proposed case study.

7.3 Applying the framework DONCIR to the case study

In this section, we present the use of DONCIR into practice to test the proper operation of its main components: *ContextGenerator, ContextManager, ContextExecutor* and *AdaptationProducer* by applying it to application of our case study illustrated with some examples of adaptation scenario. We will display all the results of our test in the console. Thus, we will display the console in the frame of our prototype to show the functioning of DONCIR in runtime.

7.3.1 Testing the functioning of the ContextGenerator

In our prototype, the launch of the application by the user is replaced by the "*run Generator*" button. In fact, for the framework DONCIR, the click on this button allows the "*Context Generator*" to manipulate the XML file named *contextModel.xml* automatically generated (see Sect. 7.2.1) to create a context instance of the application already running. The console in Fig. 17 shows the results displayed during the activation of the *ContextGenerator*.

Figure 17 shows the correct functioning of the *ContextGenerator* that succeeds to generate the instances of the context model of our case study. This is the role of the *ContextGenerator* (see Sect. 5.2.1).

7.3.2 Testing the functioning of the ContextManager

The context will be activated if the user chooses its focus. In the case of our prototype, the selection of a choice from the combo box "*focus"* and the click on the button "start," at first, allows the launch of the interface of the focus requested by the user such as "review laboratory results and radiology." Afterward, it activates the "*Context Manager*." Figure 18 shows the results displayed in the console during the activation of the *ContextManager*.

The principal role of the ContextManager is to find the dynamic context elements and assign to each one a specific controller (see Sect. 5.2.2). Figure 18 validates the functioning of the *ContextManger* of DONCIR, which succeeded to assign and activate a controller to each dynamic context element such as location, longitude and doctor category.

7.3.3 Testing the functioning of the controllers

Figure 19 describes the results appearing in the console after the activation of the controllers by the *contextManager.*

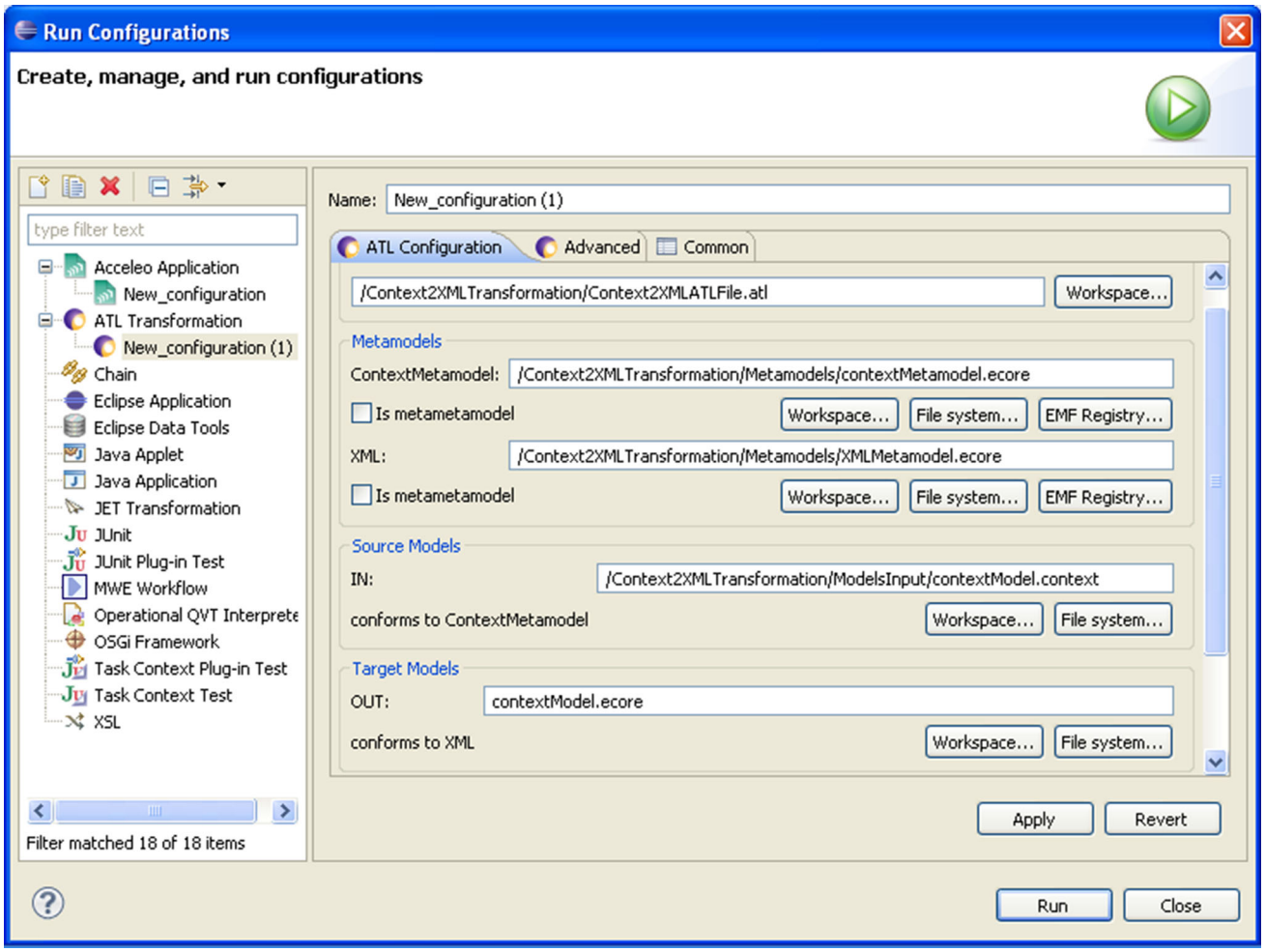


Fig. 12 ATL transformation configuration of context model of our case study

It illustrates that the controller succeeded in observing the change of the context by asking every time the new value of context property from its context provider. The console displayed the new values captured from the *contextProviders* by the controllers such as longitude, altitude and noise level.

Moreover, the controller is responsible for the detection of a context change when a property value changes and it signals a value- changed event. Figure 20 shows that when we changed the longitude value from 0 to 1500, automatically a changed value event was displayed in the console.

To sum up, Figs. 19 and 20 illustrate sample tests to show the correct functioning of the controllers of DONCIR which succeeds automatically to observe and detect the change of the context in runtime.

7.3.4 Testing the functioning of the ContextExecutor

The role of the *ContextExecutor* is to react when a controller reported that a contextElement changed. In this case, it searches all conditions depending on the context change, and if any of these conditions becomes true, it retrieves the rules it trigger and its send them to *AdaptationProducer*. To show a test of the good functioning of *ContextExecutor*, we will change the values of context elements describing two adaptation scenarios.

Figure 21 describes the following scenario of our case study: Novice Doctor Location (1200, 300, 500) that coincides in our example with roomPatient Number 100 of the patient "Rocky." The noise level is low and the light level is dark.

We observe that the change of room number causes the change of roomType and patientID since the patient properties depend on the room number. Also the "Room type" depends on the "roomNumber." Thus, the change of context property value causes the activation of conditions controllers which depend on these properties: For example, the change of *lightLevel* causes the activation of controllers

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <Root xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="XML" name="Context">
  <children xsi:type="Element" name="ContextEntity">
   <children xsi:type="Attribute" name="name" value="Doctor"/>
   <children xsi:type="Element" name="ContextProperty">
    <children xsi:type="Attribute" name="name" value="Location"/>
    <children xsi:type="Attribute" name="dataType" value="DefineType"/>
    <children xsi:type="Element" name="ContextProvider">
     <children xsi:type="Attribute" name="providerName" value="GPS"/>
     <children xsi:type="Attribute" name="className" value="ContextProviderPackage.LocationProvider"/>
    </children>
    <children xsi:type="Element" name="ContextProperty">
     <children xsi:type="Attribute" name="name" value="longitude"/>
     <children xsi:type="Attribute" name="dataType" value="Float"/>
     <children xsi:type="Element" name="ContextProvider">
      <children xsi:type="Attribute" name="providerName" value="GPS"/>
      <children xsi:type="Attribute" name="className" value="ContextProviderPackage.LongitudeProvider"/>
     </children>
    </children>
    <children xsi:type="Element" name="ContextProperty">
     <children xsi:type="Attribute" name="name" value="altitude"/>
     <children xsi:type="Attribute" name="dataType" value="Float"/>
     <children xsi:type="Element" name="ContextProvider">
      <children xsi:type="Attribute" name="providerName" value="GPS"/>
      <children xsi:type="Attribute" name="className" value="ContextProviderPackage.AltitudeProvider"/>
     </children>
```

Fig. 13 Extract of the generated target XML model during the execution of the ATL transformation

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<Context>
  <ContextEntity name = 'Doctor'>
    <ContextProperty name = 'Location' dataType = 'DefineType'>
      <ContextProvider providerName = 'GPS' className = 'ContextProviderPackage.LocationProvider'/>
      <ContextProperty name = 'longitude' dataType = 'Float'>
        <ContextProvider providerName = 'GPS' className = 'ContextProviderPackage.LongitudeProvider'/>
      </ContextProperty>
      <ContextProperty name = 'altitude' dataType = 'Float'>
        <ContextProvider providerName = 'GPS' className = 'ContextProviderPackage.AltitudeProvider'/>
      </ContextProperty>
      <ContextProperty name = 'latitude' dataType = 'Float'>
        <ContextProvider providerName = 'GPS' className = 'ContextProviderPackage.LatitudeProvider'/>
      </ContextProperty>
      <ContextProperty name = 'LocationTime' dataType = 'String'>
        <ContextProvider providerName = 'GPS' className = 'ContextProviderPackage.TimeProvider'/>
      </ContextProperty>
      <ContextProperty name = 'TimeSymbolic' dataType = 'String'>
        <ContextProvider providerName ='ContextProviderPackage.TimeSymbolicProvider' className ='
      </ContextProperty>
      <SensedAssociation name = 'locationTimeAss' refersTo ='LocationTime' type ='ContextProperty'>
        <DependAssociation name = 'altitudeAss'/>
        <DependAssociation name = 'latitudeAss'/>
        <DependAssociation name = 'longitudeAss'/>
      </SensedAssociation>
      <SensedAssociation name = 'longitudeAss' refersTo = 'longitude' type = 'ContextProperty'>
        <TemporalRelativeInterval value = '2' timeUnit = 'minute'/>
      </SensedAssociation>
      <SensedAssociation name = 'altitudeAss' refersTo = 'altitude' type = 'ContextProperty'>
        <TemporalRelativeInterval value = '2' timeUnit = 'minute'/>
      </SensedAssociation>
      <SensedAssociation name = 'latitudeAss' refersTo = 'latitude' type = 'ContextProperty'>
        <TemporalRelativeInterval value = '2' timeUnit = 'minute'/>
      </SensedAssociation>
```

Fig. 14 Extract of XMl file described in the context model of our case study

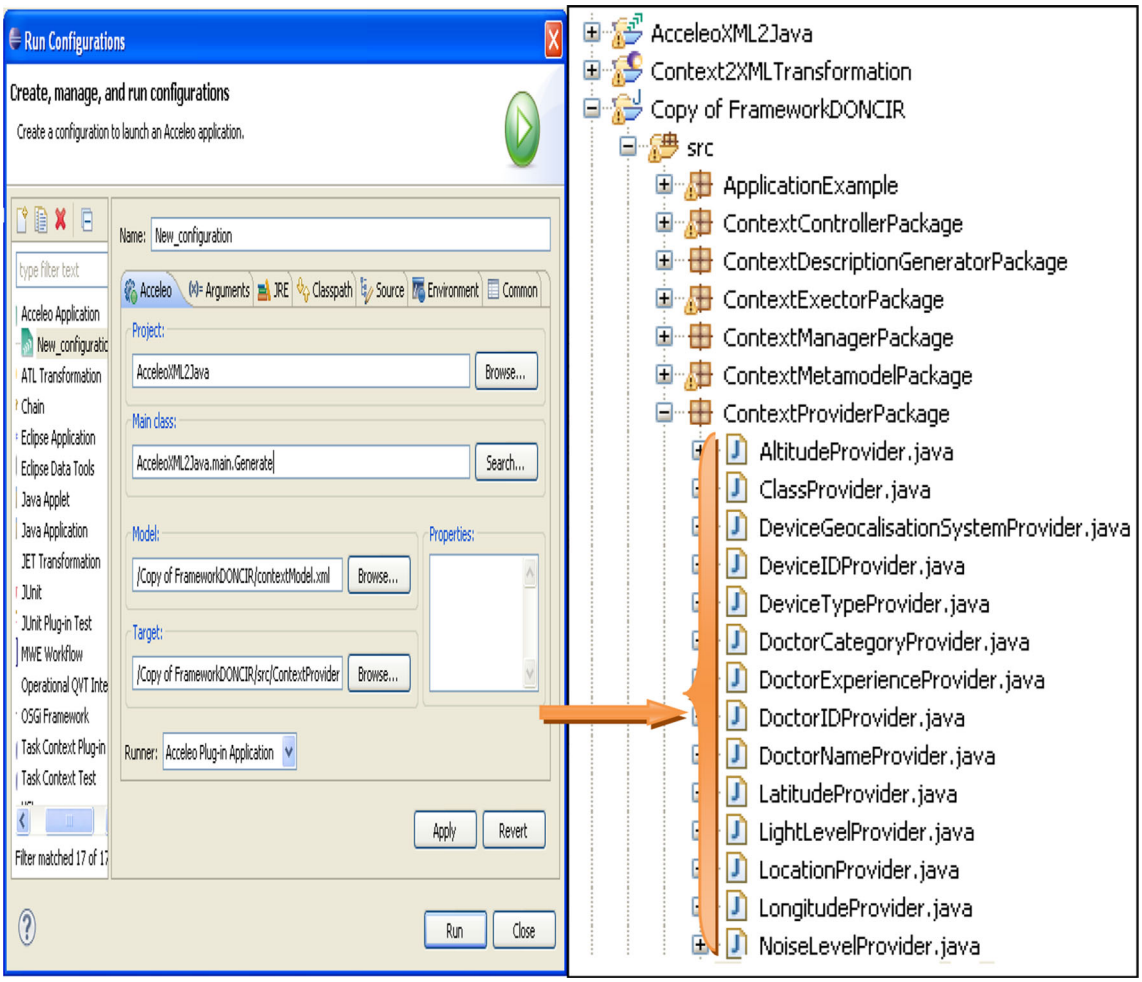


Fig. 15 Configuration for the generation of context providers' java classes

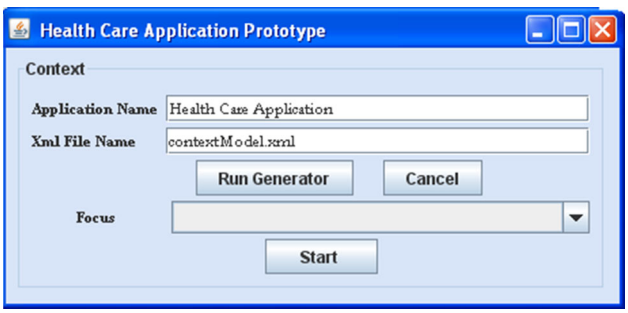


Fig. 16 Main interface of the prototype of the case study application

for the conditions Cond1, Cond2, Cond3, cond4, Cond5 and cond6.

Then, the condition Controllers calculate the new values of these conditions and compare them with the old values. If a condition changes its value to true, it will trigger a *conditionOccurred* event. Figure 22 shows that the scenario described in Fig. 21 causes the value change of cond5 (see Table 2) and consequently the trigger of *conditionOccurred* event. Thus, this figure validates the functioning of ConditionControllers. When the *contextExecutor* receives the *conditionOccurred* event triggered by the cond5, it will retrieve rules that are activated by this condition and send them to the Adaptation Producer. In the next section, we will test its functioning when receiving these rules.

7.3.5 Testing the functioning of the adaptation producer

Figure 23 shows that the rules corresponding to the Cond5 are displayed in the console. If we change the Doctor location to (2000, 500, 1000), a location that resembles to "room doctor," the adaptation rules accordingly are changed. The ring volume becomes "high"; since it is not a patient room, there is no need to reduce the volume to not disturb the patient. Figure 24 describes this scenario.

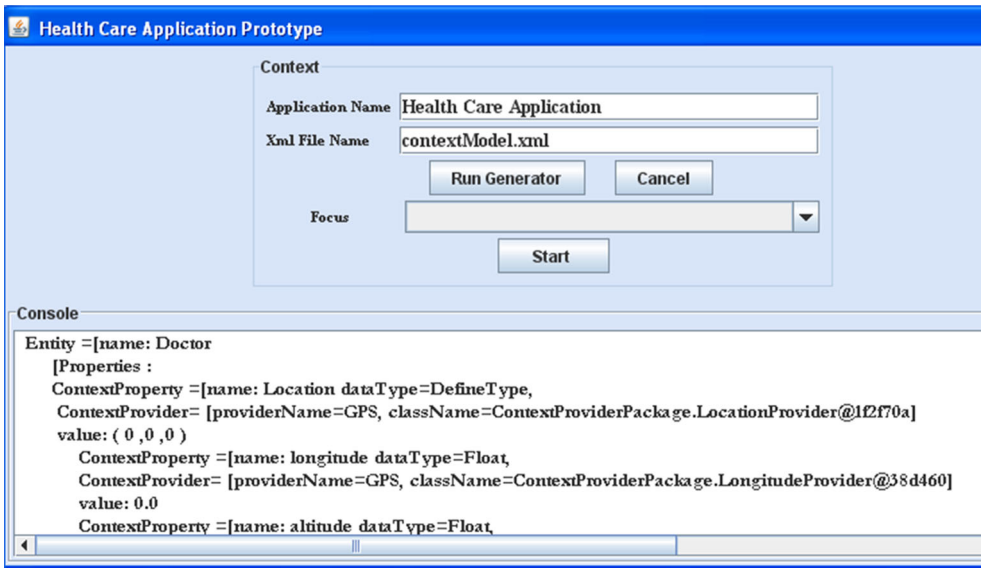


Fig. 17 Test the functioning of the ContextGenerator

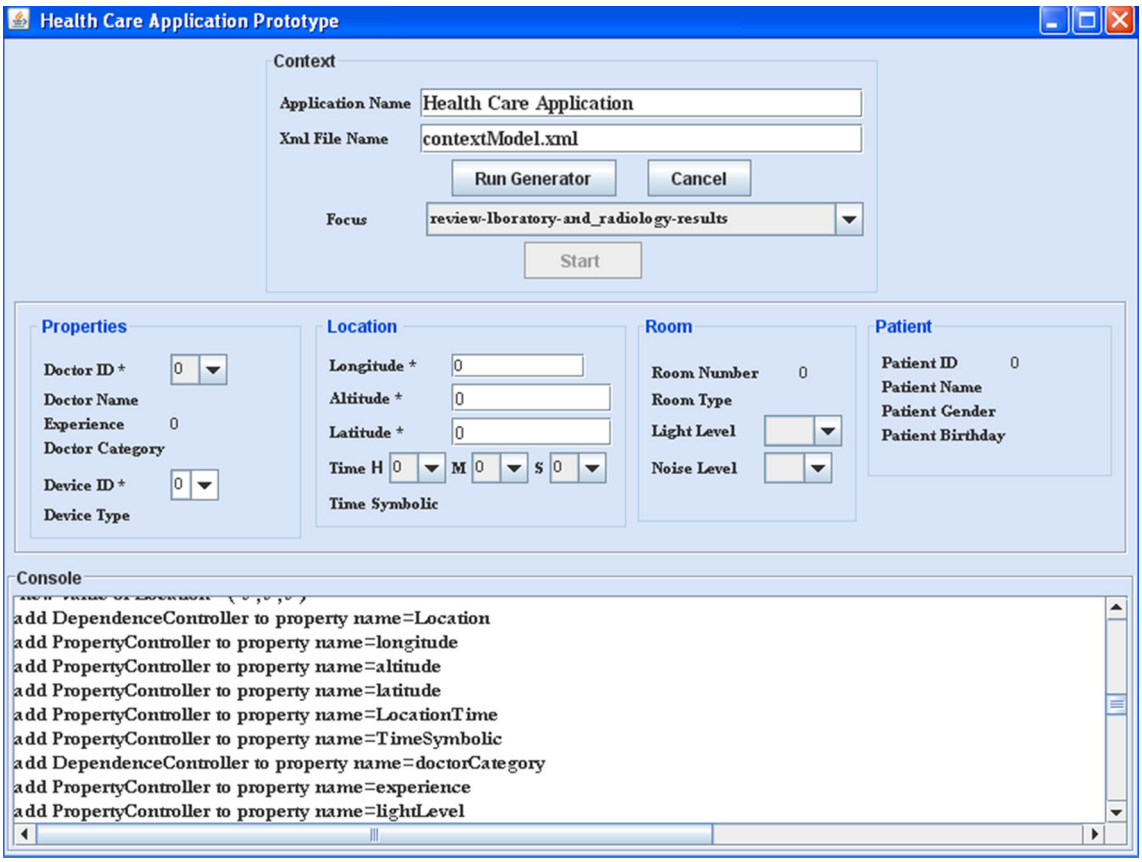


Fig. 18 Test the functioning of the *ContextManager*

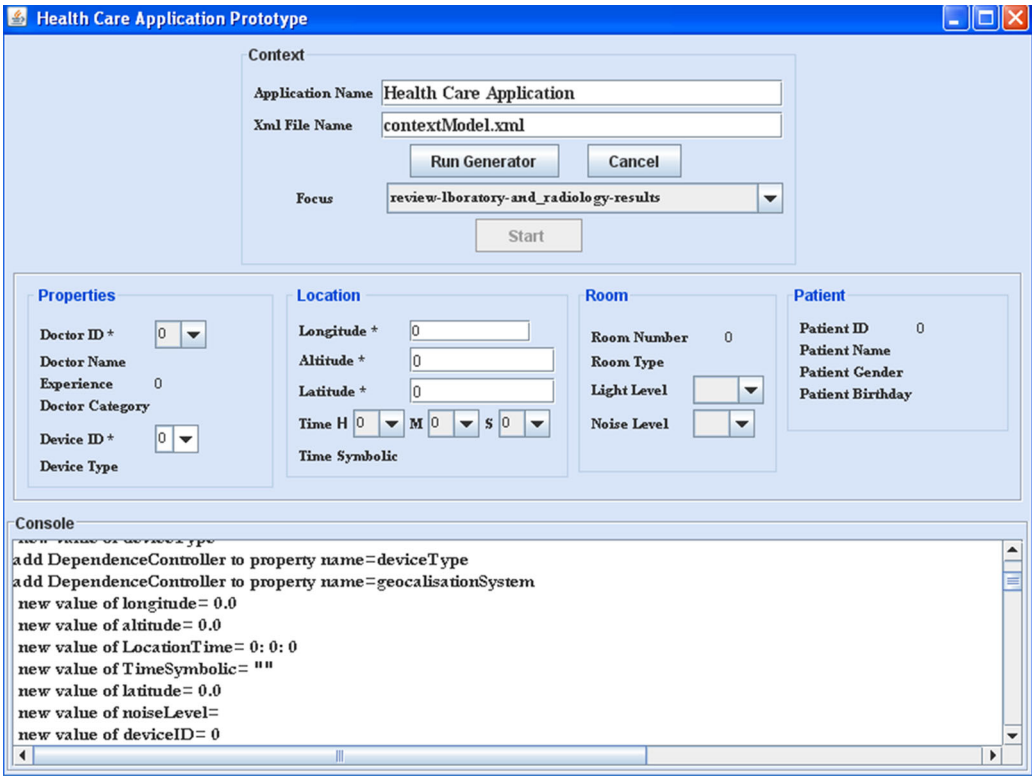


Fig. 19 Test the activation of contextProviders by the controllers

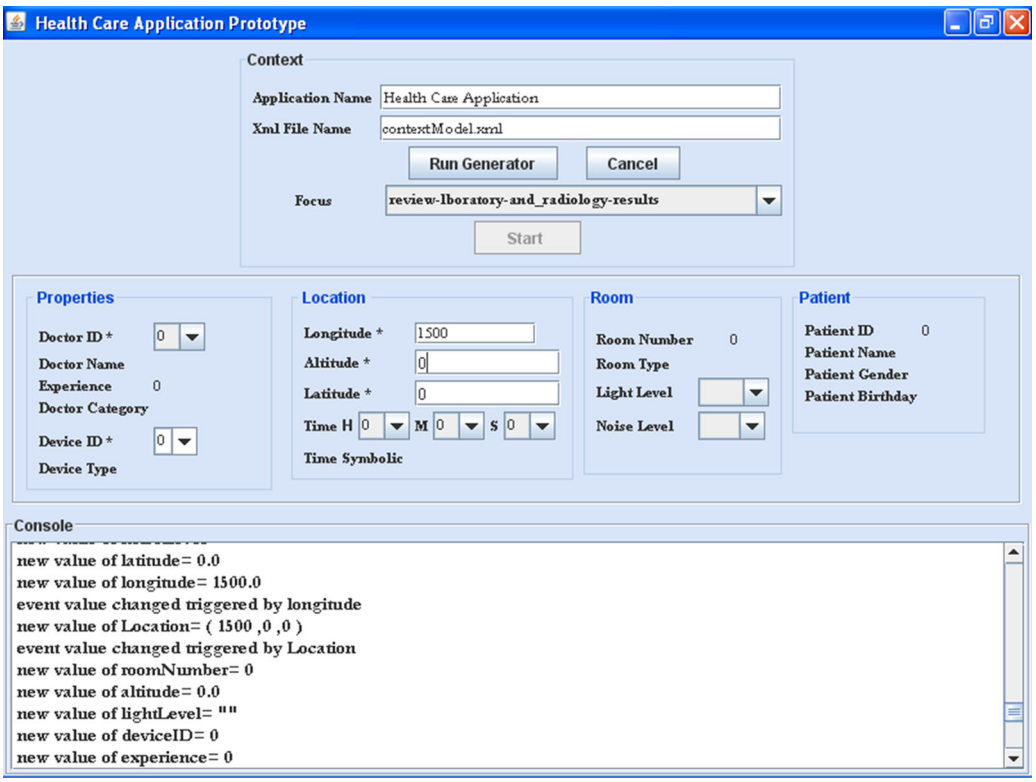


Fig. 20 Validation of the context value changed detection functioning by the controller in runtime

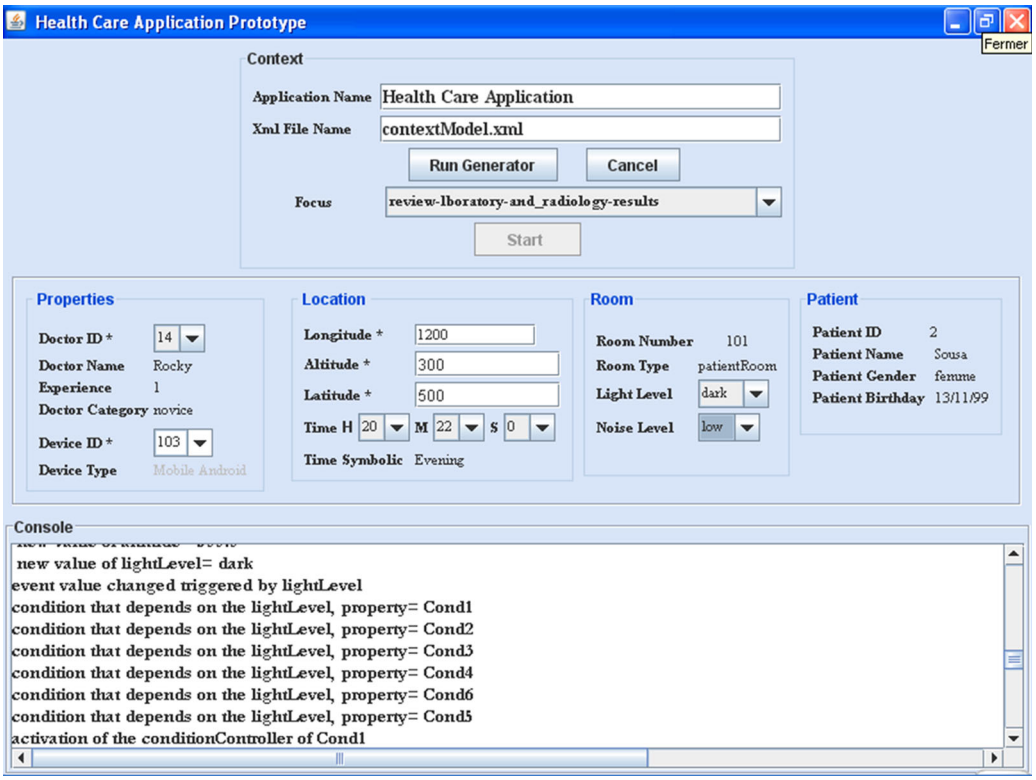


Fig. 21 The search and the activation of condition controllers by the contextExecutor

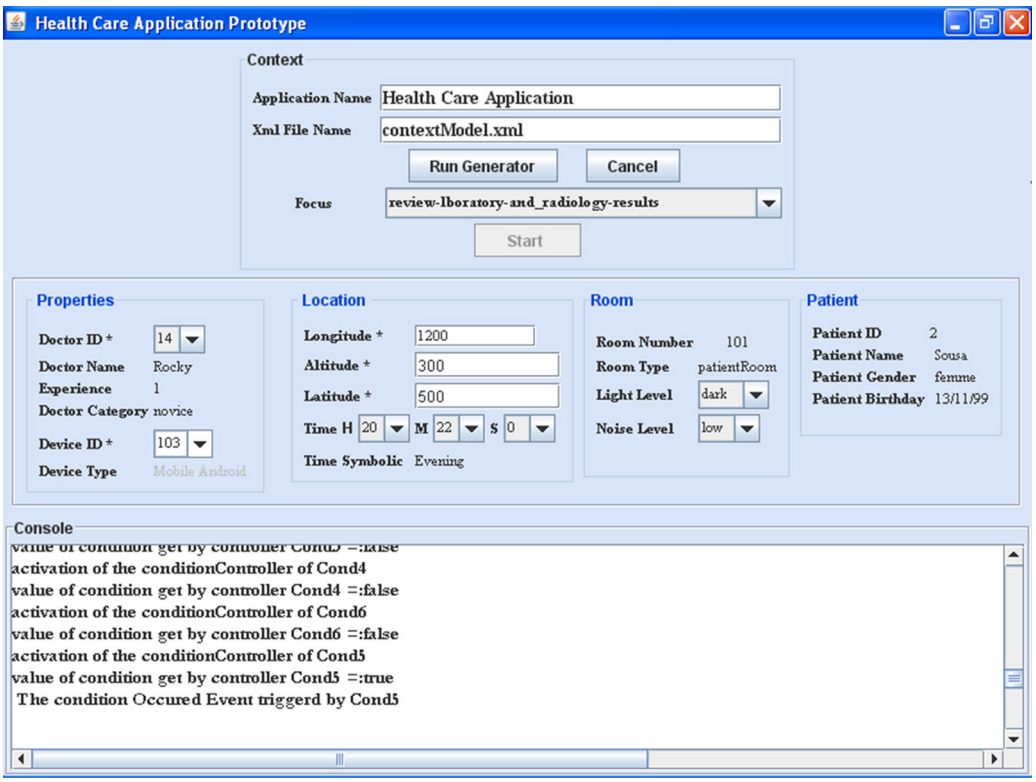


Fig. 22 Validation of condition controllers functioning

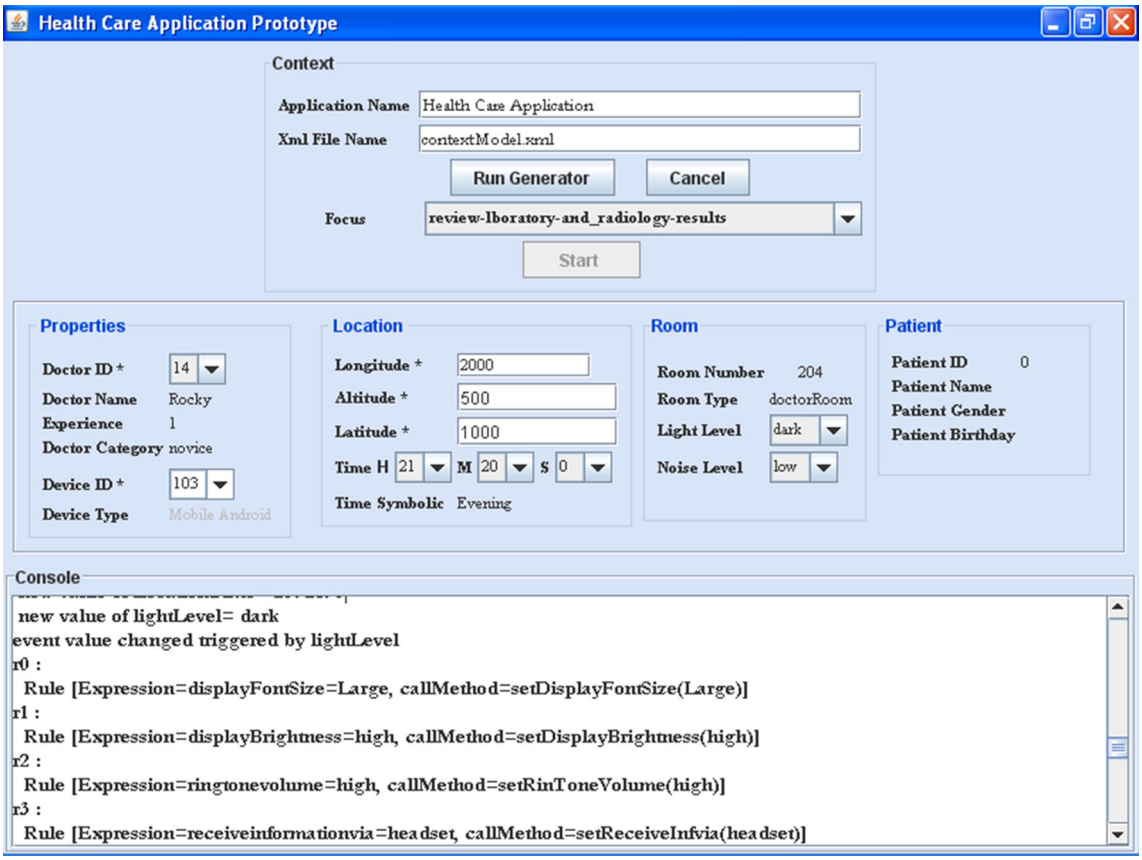


Fig. 23 Validation of adaptation scenario no

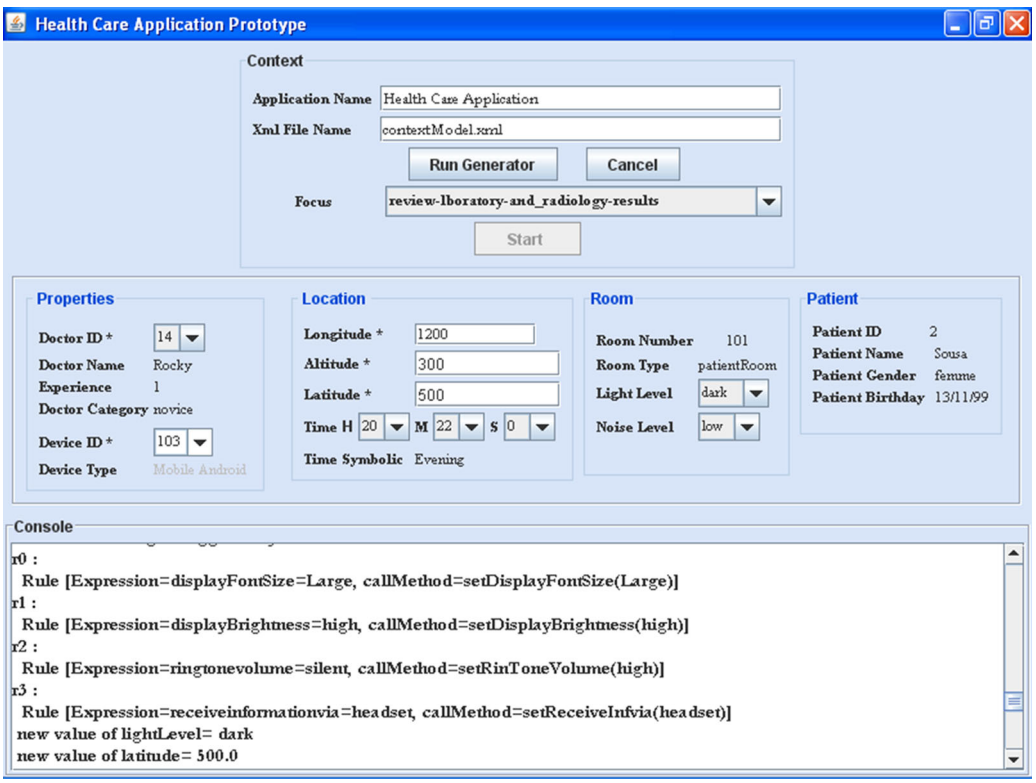


Fig. 24 Validation of adaptation scenario no 6

8 Conclusion

In this paper, we proposed CAADA, an approach for context-aware application development based on models and ECA paradigm. To validate our approach, we proposed a software framework called DONCIR that can dynamically capture, observe context changes and notify the system in runtime to perform the necessary adaptations. We have described the operating principle of DONCIR which is based on four components: *ContextGenerator*, *Context Manager, Controllers* and *Context Executor*. We presented the JAVA interface of our framework. This framework is based on generic and rich context meta-model.

To help developers use our framework DONCIR, we presented a model-driven approach strictly based on the paradigm MDA that provides the capability to automatically generate XML file descriptive of context model. The usage of this approach enables the developer to define a context model using a comprehensive visual representation. Furthermore, it supports the context model validity and conformity to our meta-model. Otherwise, we proposed a transformation model to code based on meta-model XML and XML file generated to create java classes responsible for capturing and providing context element values. All these steps are performed in the same eclipse plug-in and using configuration mechanisms to reduce programmer effort and to help him to develop its application. We illustrated the feasibility of DONCIR in runtime through a set of testing scenarios on a case study in the healthcare domain. The design results showed that DONCIR is easy use for context-aware application development. The herein presented testing results allowed us to show the feasibility of all functionalities of DONCIR, namely automatic generation of an instance of the context from the XML file, context capturing, control of dynamic context elements, and context change detection and triggering adaptation rules.

Our work has the following merits: a rich, domain-independent MOF compliant context meta-model that provides the specification of all context-relevant aspects (structural, dynamic, behavioral); the framework DONCIR including its runtime infrastructure and its programming model which provides observing the context and adapting the application to context changes in runtime; an comprehensive application development process guided by mechanisms that support the rapidity of the development task and that do not require much programming and configuration efforts.

In conclusion, CAADA is original in the sense that combines a generic and rich context meta-model, a full and rapid process development and a support of implementation.

In our ongoing research and development efforts, we plan to further apply this approach in the development of large-scale context-aware applications in different application domains. This will delimit the relevance of our framework in other areas and validate its genericity and reuse advantages. We also plan to offer reliable solutions for testing the effectiveness of our approach using complex conditions and information deduced from the context history.

References

1. Achilleos, A., Yanga, K., Georgalas, N.: Context modelling and a context-aware framework for pervasive service creation: a model-driven approach. J. Perv. Mob. Comput. **6**(2), 281–296 (2010)
2. ATLAS group LINA and INRIA Nantes: ATL: atlas transformation language specification of the ATL virtual machine. Version 0.1 (2005)
3. Bardram, J.E: The Java context awareness framework (JCAF)—a service infrastructure and programming framework for context-aware applications. In: Proceeding of the Third International Conference on Pervasive Computing (Pervaive'2005), pp. 98–115. Munich (2005)
4. Bezivin, J.: Towards a precise definition of the OMG/MDA framework. In: 16th Annual International Conference on Automated Software Engineering (ASE 2001), pp. 273–280. San Diego (2001)
5. Costa, P.D.: Architectural support for context-aware applications—from context models to services platforms. Ph.D. Thesis, Enschede (2007)
6. David, P.C., Ledoux, T.: WildCAT: a generic framework for context-aware applications. In: Proceeding MPAC'05 Proceedings of the 3rd International Workshop on Middleware for Pervasive and Ad-hoc Computing, pp. 1–7. ACM, New York (2005)
7. Dey, A.K., Salber, D., Futakawa, M., Abowd, G.D.: An architecture to support context-aware applications. GVU Technical Report GIT-GVU-99-23. In: The 12th Annual ACM Symposium on User Interface Software and Technology (UIST '99) (1999)
8. Dey, A.K., Abowd, G.D.: Towards a better understanding of context and context-awareness. In: The Workshop on The What, Who, Where, When, and How of Context-Awareness, As Part of The: Conference on Human Factors in Computing Systems (CHI 2000). The Hague (2000)
9. Dey, A.K., Abowd, G.D., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Hum. Comput. Interact. J. **16**(2), 97–166 (2001)
10. Helming, J., Koegel, M.: What every eclipse developer should know about EMF. Eclipse Source (2015)
11. Henricksen, K., Indulska, J.: Developing context-aware pervasive computing applications: models and approach. J. Perv. Mob. Comput. **2**(1), 37–64 (2006)
12. Jaouadi, I., Ben Djemaa, R., BenAbdallah, H.: A generic meta-model for context-aware applications. In: The 23 International Conference on Systems Engineering (ICSEng 2014), pp. 587-59419-21. Las vegas (2014)
13. Motti, V.G., Vanderdonckt, J.: A computational framework for context-aware adaptation of user interfaces. In: IEEE Seventh International Conference on Research Challenges in Information Science (RCIS), pp. 1–12. Paris (2013)
14. Object Management group: Meta object facility (MOF) specification. OMG Document, version 1.3 (2000)
15. Object Management group: The model driven architecture. Mars (2015)
16. Pham, H.N., Mahmoud, Q.H., Ferworn, A., Sadeghian, A.: Applying model-driven development to pervasive system engineering. In: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments (ICSE, 2007), p. 7 (2007)
17. Schilit, B.N., Adams, N., Want, R.: Context-aware computing applications. In: Proceeding WMCSA'94 Proceedings of the 1994

First Workshop on Mobile Computing Systems and Applications, pp. 85–90. IEEE Computer Society, Washington, DC (1994)
18. Santos, L.O.S., Wijnen, R.P.V., Vink, P.: A service oriented middleware for context-aware applications. In: Proceeding of 5th International Workshop on Middleware for Pervasive and Ad-hoc Computing: Held at the ACM/IFIP/USENIX 8th International Middleware Conference, pp. 37–42. ACM, New York (2007)
19. Serral, E., Valderas, P., Pelechano, V.: A model driven development method for developing context-aware pervasive systems. In: Proceeding 5th International Conference, Ubiquitous Intelligence and Computing (UIC 2008), pp. 662–676. Oslo (2008)
20. Vale, S., Hammoudi, S.: COMODE: a framework for the development of context-aware applications in the context of MDE. In: Fourth International Conference on Internet and Web Applications and Services (ICIW '09), pp. 261–266. Venice/Mestre (2009)
21. Vieira, V., Tedesco, P., Salgado, A.C.: Designing context-sensitive systems: an integrated approach. J. Exp. Syst. Appl. Intell. Collab. Des. **38**(2), 1119–1138 (2011)
22. MarkLogic Server: XQuery and XSLT reference guide. MarkLogic (2015)
23. Zimmermann, A., Lorenz, A., Oppermann, R.: An operational definition of context. In: Modeling and Using Context, 6th International and Interdisciplinary Conference (CONTEXT 2007), Computer Science, pp. 558–571, volume 4635. Springer, Berlin (2007)

Imen Jaouadi received her engineering diploma from Sousse University, Tunisia, in 2007. Then she obtained her Master Degree in Intelligent Information system from Kairouan University, Tunisia, in 2010. She worked at the University of Monastir, Tunisia, from 2010 until 2014. Actually, she is a Ph.D. student in laboratory Multimedia, Information Systems and Advanced Computing (MIRACL) of Sfax University. Her research interests include software engineering, human–computer interaction, adaptation of interactive systems and context-aware systems. She has participated in several international conferences.

Raoudha Ben Djemaa received her Master degree in Information system and New Technology from Sfax University, Tunisia, in 2002. She obtained her Ph.D. in informatics from Sfax University, Tunisia, in April 2009. She is actually an assistant professor in the Higher Institute of Computer Sciences and Technology Communication of Hammam Sousse in Tunisia. She is also a member of the laboratory Multimedia, Information Systems and Advanced Computing (MIRACL) of Sfax University, Tunisia. Her research interests include software engineering, methodologies and approaches for adaptive web applications, development and approaches for adaptive web services and finally IHM adaptation. She has participated in several national and international conferences.

Hanêne Ben-Abdallah received a B.S. in Computer Science and B.S. in Mathematics from the University of Minnesota, MPLS, MN, a MSE and Ph.D. in Computer and Information Science from the University of Pennsylvania, Philadelphia, PA. She worked at the University of Sfax, Tunisia from 1997 until 2013. She is now full professor at the Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Kingdom of Saudi Arabia. She is a member of the Mir@cl laboratory, University of Sfax. Her research interests include software design quality and reuse techniques applied to data warehouses, business processes and web-based applications. She has published over 60 papers in refereed journals, international conferences and book chapters in these research areas.