Electronic Communications of the EASST
Volume 28 (2010)

Proceedings of the Third International DisCoTec Workshop on Context-Aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS 2010)

MLContext: A Context-Modeling Language for Context-Aware Systems

José R. Hoyos, Jesús García-Molina and Juan A. Botía

14 pages

Guest Editors: Sonia Ben Mokhtar, Romain Rouvoy, Michael Wagner
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

MLContext: A Context-Modeling Language for Context-Aware Systems

José R. Hoyos[1], **Jesús García-Molina**[1] **and Juan A. Botía**[2]

[1] Dpt. Informática y Sistemas, [2] Dpt. Ingeniería de la Información y las Comunicaciones
Universidad de Murcia, Campus Espinardo, 30100 Murcia, Spain

Abstract: Context awareness refers to systems that can both sense and react based on their environment. The complexity of these systems makes necessary to apply software engineering techniques in their development, such as Model-Driven Software development (MDD). One of the main difficulties that developers of context-aware systems must tackle is how to manage the needed context information. In this paper, we present MLContext, a textual Domain Specific Language (DSL) which is specially tailored for modeling context information and automatically generating software artefacts from context models. It has been designed to provide a high-level abstraction, to be an easy to learn, and to promote reuse of context models. We have built a toolkit including an editor and a parser to convert MLContext textual specifications into models. As a proof of concept, we have automatically generated ontologies and Java code for the OCP middleware. MLContext models can be reused in applications with the same context because they do not include details related to the platforms or the implementation. These context models can be specified by non-developers users because MLContext provides high-level abstractions of the domain.

Keywords: MDD, Domain Specific Language, Context Modeling, Context-aware

1 Introduction

Context awareness is an essential aspect of pervasive computing (also termed as ubiquitous computing). Context-aware systems are able to quickly adapt their behavior to changes in context without explicit user intervention. The context information needed may be obtained from a variety of sources, such as networks, devices, or by applying sensors and browsing user profiles. Since the building of context-aware systems is an intricate task, software engineering techniques for tackling its complexity have been used, such as modeling or frameworks based solutions. Several architectures for distributed context-aware frameworks have been developed [KMK+03] [BDR07]. These provide a generic infrastructure with middleware and context managing services. Furthermore, as context storing and processing play a key role in context-aware systems, a number of context modeling approaches have been proposed, which differ in the data structures used to represent the context information [SAW94] [PB05]. However, as in other application domains, the activity of modeling in context-aware systems is currently directed towards Model-Driven Development (MDD). Several approaches have recently been put forward to take advantage of MDD techniques in the construction of context-aware systems and pervasive applications [ADB07] [PMFS07], but a great research effort is still needed to provide appropriate

languages, methods and tools to efficiently support the model-based development of this kind of applications. While most of the proposed approaches have defined a UML profile with which to model context, DSLs created from scratch are currently considered to be more appropriate in most situations [KP09]. UML Profiles is not a first-class extension mechanism because it does not allow for modifying the UML metamodel, and the new elements, which specialize existing elements, must not violate the abstract syntax rules and semantics of UML. Therefore, UML profiles should be restricted to define modeling languages whose abstract syntax and semantics is close to those provided by UML [Sel07]. In this respect, in this paper we present a textual DSL, named MLContext, which has been specially tailored to model context information. This language has been designed with three main requirements in mind: i) to provide a high-level abstraction for building platform independent models, ii) to be a language which is simple and easy to learn, so that final user can write and understand context models, iii) to promote the reuse of context models in different context-aware applications, by separating the implementation dependent aspects from the domain aspects. MLContext models can be used to automatically generate context management related software artifacts for any context-aware middleware. As a proof of concept, MLContext has been applied to generate code for the OCP middleware [BVP+09].

MLContext is a component of an MDD solution we are developing for context-aware systems and, to the extent of our knowledge, it is the first approach that separates the definition of the context elements from the details related to the context sources and application dependent aspects. This document is organized as follows. In Section 2, the main issues related to the representation of context are analyzed and the design choices to be taken into account when designing a DSL for modeling context are then identified. In Section 3, MLContext abstract syntax and notation are presented through an example. Section 4 shows how OCP code can be automatically generated from MLContext models. In Section 5, a survey of current related work is presented. Finally, Section 6, shows our conclusions and future work.

2 Domain study and desing decisions

The main phases in the development of a domain specific language are discussed in [MHS05]. The first phase is the domain analysis which aims to identify and describe the domain concepts and their properties. To gather the domain knowledge, we have analyzed the most relevant approaches and frameworks dealing with context-awareness and context representation and we have also considered the knowledge gained in creating and using the OCP middleware. Next, we show the conclusions of the domain analysis that led us to the requirements of the DSL to be created. In building a DSL for modeling context, one of the main obstacles is the lack of a commonly accepted definition of what context really is. Dozens of context definitions can be found in literature, each of which differs in what information is actually part of a context. In [DA00], context information is defined as ”any information that can be used to characterize the situation of any entity”. An entity could be a person, a place, a physical or abstract object, or an event that is considered to be relevant to the interaction between a user and an application, including the user himself. This idea of linking context information to an entity or subject is a key aspect of our approach. When entities are modeled, they can be classified in a hierarchy of categories of entities. For example, in the context of a hospital, patients and medics are persons,

and medics are also hospital employees.

We must take into account that the information which is part of a context may come from several sources, which may use different formats to represent the information they provide. As indicated in [HIR02], there is usually a significant gap between sensor output and the level of information that is useful to applications, and this gap may be bridged by various types of context information processing. For example, a location sensor may supply raw coordinates, whereas an application might be interested in the identity of the building or room a user is in. Applications usually represent the context information as a set of designer-defined descriptors (e.g. numerical values of time or device IDs). While this format might be appropriate for developers, it could be difficult for the users (who are used to handling higher-level semantic descriptors) to understand.

When modeling context, it is necessary to distinguish among different types of context information, so that a taxonomy is useful to appropriately model the concepts. However, most existing approaches do not define such a taxonomy. Instead, they use a "generic" context, no matter what type of context information they are dealing with ([ADB07][HIR02][VK07][SB05]). Some taxonomies have been proposed such as [GM02], [AFG+07], [PSM+05], [Sch05], and when they are compared, the following is found. Some types of context, such as the social context or the physical context, are common to nearly all the proposals, while others like the computational context only appear in some of them. This is because most of these classifications are focused on specific domains. For example, Ardissono et al. [AFG+07], make their classification by bearing in mind the management of context-aware web service-based workflow systems, and Schmidt [Sch05] only considers contexts that are relative to e-learning systems in order to consider the learners situation in an appropriate manner. Moreover, the same information is considered to be of different context types, such us the role of a user in [GM02] and [PSM+05], or the bandwidth communication cost in [GM02] and [PSM+05].

A context model should normally include information on temporal aspects. In this regard, context information can be characterized as being static or dynamic. The value of static information remains unchanged over time whereas the value of dynamic information may vary depending on the time instant we request it. Two important observations are made in [HIR02]. The first is that "context information is imperfect" and hence the system must therefore assure information reliability and availability. This does not directly affect the task of modeling the context, but some kind of quality measurement should be supported by a DSL designed to model contexts in order to validate the accuracy of the information supplied. The second observation is that "context information is highly interrelated" because complex information is obtained from one or more simple contexts. Furthermore, the context of an entity often refers other entities contexts (for example, an entity formed of other entities). Other features of a context model such as the user profile and the context history [HSKK09] are normally considered to be optional. The user profile is a collection of personal data associated with a specific user. This information can be exploited by systems by taking into account the users characteristics and preferences. Context history becomes necessary when information is measured over time and it is necessary to keep a trace of its values in the time dimension. Our analysis, has allowed us to identify the following requirements for the MLContext language:

1. **High-level abstraction**. The language should provide a high-level abstraction by means of constructs that are close to the domain concepts (e.g. entity, context or type of context).

This would thus encourage users who are not developers to participate in context modeling.
2. **Platform Independent**. The language should allow the creation of platform independent models and free the user from having to provide implementation details.
3. **Application domain independent**. Since contexts have many similarities in different context-aware applications, MLContext should allow contexts to be modeled regardless of the context-aware domain.
4. **Types of context**. The language should support the ability to model different types of context, such us physical, social or computational context, because this is closer to reality. Moreover, separating information in multiple contexts can be exploited to achieve improvements such as more efficient storage and retrieval methods.
5. **Traceability, quality and temporal mechanisms**. The language should provide a traceability mechanism to support historic information, a mechanism for measuring the quality of the supplied information, and it should support both static and dynamic temporal information.
6. **Model reuse**. The language should promote the reuse of models, signifying that context models could be reused in different applications using the same context.
7. **Usable language**. Finally, the language should be easy and intuitive to use.

In order to satisfy these requirements, we have made some design decisions, which are discussed below. The three main concepts around which the definition of the language are organized are entity, context and source of context, and a context can also be composed of other related contexts. Unlike other approaches, the modeling in MLContext is *centered on entities* rather than categories. That is, in the specification of a context, we do not model categories of entities (e.g. a patient) but rather specific instances (e.g. patient 'Burt Holmes'). This is mainly because context sources are linked to entities rather than categories. In fact, two entities from the same category may have different context sources for the same property. Another reason is that the developer must give detailed information about each entity when instances of a category are created. MLContext can infer the category based on the properties of the entities belonging to it. Changes in the properties of the entities will be automatically propagated to their categories. The developer does not, therefore, have to specify categories, which saves time and effort.

It may sometimes be necessary to explicitly specify a category, as when dealing with legacy systems in which the number of entities may be very high and the information about the entities can be obtained from a database or repository to create the instances. In these situations, MLContext permits the developer to specify a *generic entity*: an entity with no specific values of its attributes, from which the category is generated.

The types of contexts have been organized in a context taxonomy which includes the most commonly used types: *physical, environment, computational, personal, social* and *task*. The context of an entity is generally composed of several related contexts of different types, and a distinction can therefore be made between the complex context (the *parent* context) and simple contexts (the *child* contexts) of which it is composed. Simple contexts are formed of a set of contextual information of the same type and can refer to other entities. The information gathered in simple contexts may come from different sources. In MLContext models, the source descriptions are separated elements in order to offer users a high level of abstraction. The language allows us to express whether the information is static or dynamic, whether or not it must be registered in the trace, or the accuracy of the information supplied by the sources.

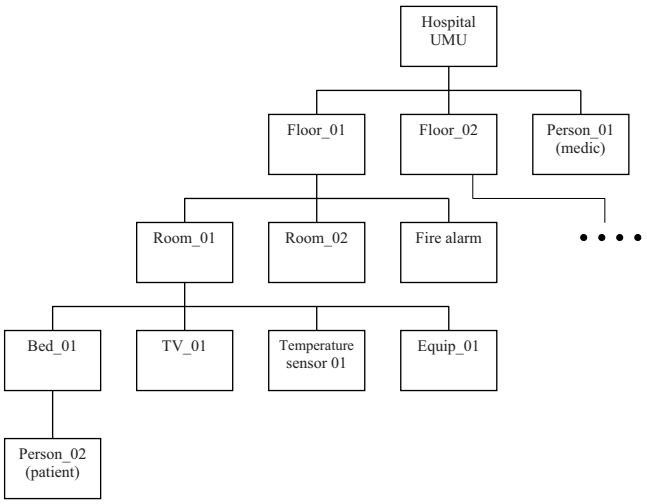


Figure 1: Aggregation hierarchy of the UMU hospital simplified example.

MLContext models do not contain platform or application dependent details. Nevertheless it is possible to generate code related to the storage and retrieval of the context information. Moreover this favors context model reuse in different context-aware applications. For example, a complete and detailed context model of an art museum could include information about the location of surveillance cameras, presence detectors and fire detectors. It could also include the spatial distribution of rooms, location of emergency exits and a complete profile of every art object on display in the museum. By applying MLContext, the same context model could be used in different applications as: i) A surveillance system, which can trigger an alarm if an intruder is detected at night. ii) An emergency system, which can detect the presence of fire and guide the people to the nearest emergency exit. iii) A guided-tour system, which can guide visitors through the museum and explain each object of art they are looking at.

3 A DSL for modeling context

A DSL, normally, consists of three elements: abstract syntax, concrete syntax, and semantics. The abstract syntax defines the concepts of which the DSL consists and the relationships between them, and also includes the rules which constrain how the models can be created. In MDD, the abstract syntax of a DSL is specified in a metamodel, which is created by using metamodeling languages (e.g. Ecore). The concrete syntax defines a notation for the abstract syntax, and semantics is normally provided by means of model transformations which transform a model expressed in the DSL into models expressed in languages with well-defined semantics (e.g. a programming language). In this section we present the MLContext DSL, which has been built according to the requirements and design choices discussed in the previous section.

We have chosen an example based on a simplified case study of a hospital, in which doctors, patients and the hospitals equipment must be taken into consideration to represent the context of the application. In this scenario, a typical ubiquitous service quickly allows the medic to

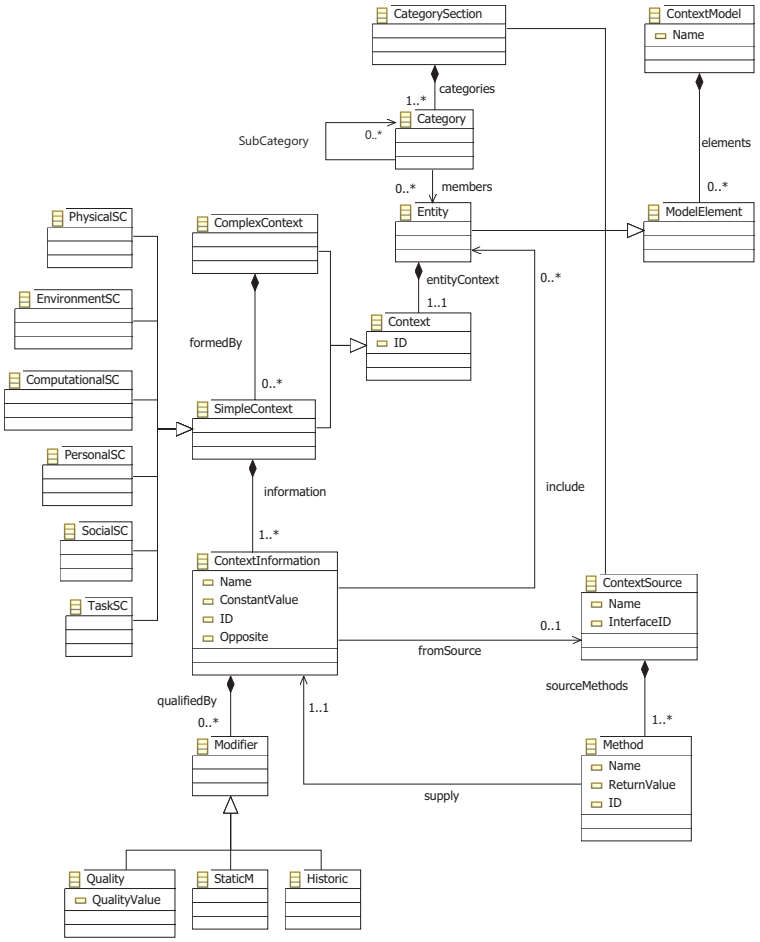


Figure 2: MLContext's abstract syntax metamodel.

automatically access information about a patient he is dealing with by using both medic and patient context location. Figure 1 shows an excerpt of the aggregation hierarchy of the hospital which must be born in mind when creating the context model. As can be observed, the root entity is the hospital itself which is formed of floors and people. Each floor is formed of rooms and has a fire alarm with a fire presence detector. For the sake of simplicity, we have represented only two floors and two rooms. *Room 01* contains a bed, a television, a temperature sensor and equipment, which can be attached to a patient in order to monitoring his/her vital signs.

The MLContext metamodel is shown in Figure 2. It has been designed around the concept of entity (*Entity metaclass*), and all contextual information always refers to an entity as explained above. A complex context (*ComplexContext metaclass*) is an aggregation of simple contexts (*SimpleContext metaclass*) of different types (e.g. physical and social). For example, the context of a person can be an aggregation of a personal context (e.g. name and age) and a physical context (e.g. temperature). A simple context is formed of one or more pieces of context information (*ContextInformation metaclass*) of the same type (e.g. name and age are personal information).

The context information of a simple context can contain a reference to other entities of the model (e.g. the complex context of one floor of the hospital can contain a simple context with environmental information referring to the rooms it is composed of). This is represented by the *include* relationship in the metamodel. The value of a *ContextInformation* can be a constant or can come from a source of context (*ContextSource metaclass*) as indicated in the *fromSource* relationship. The information from the source is supplied through a method (*Method metaclass*) from the physical device interface. The same source can supply context information to several entities and can use a different method for each of them. The *Modifier* hierarchy represents the three optional modifiers for the context information: quality (*Quality metaclass*) to specify the accuracy of the information, historic (*Historic metaclass*) to keep trace of the information, and static (*StaticM metaclass*) to specify that the information does not change over time. The *CategorySection metaclass* represents the set of categories of the model. Each category (*Category metaclass*) contains all the entities belonging to that category, and can also be a subcategory. An entity or subcategory can belong to more than one category (i.e. multiple classification). MLContexts abstract syntax has been modeled using Ecore metamodel language supported by the Modeling Framework (EMF) [EMF10].

To create an MLContext model, a declaration must be written for each model element. Since the *hospital_UMU* entity is formed of two floor entities, the context information of the hospital must include the context information of each of its floors. The following declaration shows that *hospital_UMU* has a complex context formed of an environment context, which only has a "contains" property, which expresses that the hospital contains two floors (*floor_01* and *floor_02*). These entities must be specified later in the model.

```
entity hospital_UMU context {
     environment {
          "contains" : floor_01, floor_02 static }}
```

The modifier static indicates that this information does not change its value. Note that medics are not included in the declaration, as is suggested by the aggregation hierarchy, because they can change their locations from one room to another but they are in the model. Each floor could be specified as follows.

```
entity floor_01 context {
     environment {
          "contains" : room_01, room_02 static }
     physical {
          "fire_presence" source fire_alarm }}
```

The specification of the *floor_01* entity shows a complex context formed of two simple contexts of the *environment* and *physical* types. The environment information expresses that the floor contains two rooms and the physical information expresses that it has a "*fire_presence*" property. The *source* keyword indicates that the value of this property comes from a source of context named *fire_alarm*. It is now necessary to model each of the rooms in the hospital.

```
entity room_01 context {
     environment {
          "contains" : bed_01, tv_01 static }
     physical {
          "temperature" source temp_01 }}
```

In the *room_01*, there are a bed named *bed_01* and a television named *tv_01*. We are interested in knowing the temperature of the room in order to activate the central heating if the room becomes too cold. This is also context information of a physical type that we can obtain from a source named *temp_01*. Now, let us suppose that there is a patient *person_02* in *bed_01*:

```
entity bed_01 context {
     environment {
          "contains" : person_02 }}
```

The *person_02* entity has a lot of context information which is useful in our modeling example. This entity would have a complex context formed of the aggregation of personal, social and physical contexts.

```
entity person_02 context {
     personal {
           "name" :     "John"  static
           "surname" : "Smith" static
           "age" :      "39" }
     social {
           "role" :     "patient" }
     physical {
           "temperature" source equip_01 historic
           "pulse" source equip_01 }}
```

The *personal* context refers to the profile of the patient which is context information of the personal type. Note that there is no context source associated with the name, surname or age. This is considered to be information provided by the user. As indicated in the *social* context information section, the "*role*" this person is playing in the hospital is "patient". Other people can play other roles such us "visitor" or "medic". To monitor the state of the patient, we need to know his "*temperature*" and "*pulse*". This information is enclosed in a physical context. We assume that the patient is connected to an *equip_01* which can provide us with the required context information. Note that, in this case, we associate the same source with two different pieces of physical context data. While the patient is in the *bed_01*, the source *equip_01* could be associated with the physical equipment in room *room_01* containing this bed, but if the patient is moved to *room_02* the source *equip_01* will be attached to the physical monitor of this new room. We have specified that temperature is context information of the historic type because we are interested in keeping a trace of its values over the time.

Once we have modeled all the entities, we have to specify the context sources. These correspond with the source element in order to establish a match between the source and its physical device. One example of a source definition is:

```
contextSource equip_01 {
     interfaceID : "AdapterX670"
         person_02.temperature {
             methodName : "getTemp()"
             returnValue : "float" }
         person_02.pulse {
             methodName : "getPulse()"
             returnValue : "float" }}
```

The *interfaceID* keyword refers to the identification of the physical device adapter (i.e. an X10

standard address), which will provide us with the desired information. This source provides values for the temperature and pulse properties of the *person_02* entity. The *methodName* keyword refers to the method to be invoked to retrieve information from the adapter, and the *returnValue* to the type of the information returned. These methods provide the interface for accessing the source independently of the actual interface of the source (e.g. the *Adapter* design pattern is applied). In order to specify the categories to which entities belong, it is necessary to add a *categories* element to our model as follows:

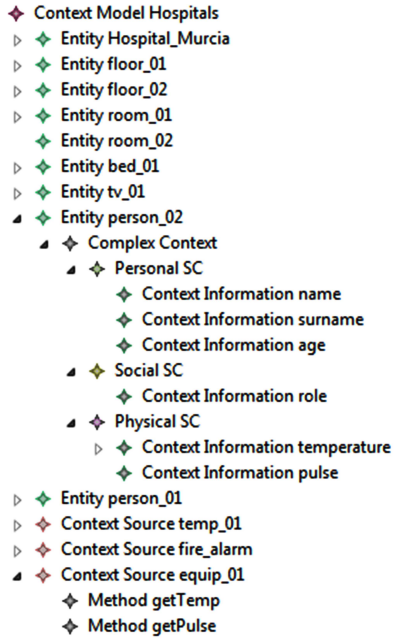
```
categories {
        Hospital : hospital_UMU
        Floor : floor_01, floor_02
        Room : room_01, room_02
        Bed : bed_01
        Person : person_02, Medic
        Medic : person_01 }
```

The aforementioned definition states that the *hospital_UMU* entity belongs to the *Hospital* category, the *floor_01* and *floor_02* entities belongs to the *Floor* category, and so on. A category can have subcategories resulting in a category hierarchy. In the previous example the *Medic* category is a subcategory of the *Person* category. We do not need to specify the properties for each category because MLContext can infer them from its entities (the union of the properties of the entities belonging to that category). MLContext takes into account the properties inherited from other categories (for example the *Medic* category will have only the specific properties not included on the *Person* category because *Medic* is a subcategory of *Person* and the "*name*", "*surname*" and "*role*" properties are inherited from it). In the case of a property name collision, a warning is shown so that the user can make the appropriate decisions. A property in a category indicates that a member of that category could exhibit the property, but this is not mandatory.

When an MLContext model is compiled an executed to generate software artifacts, if the designer has not specified a category for some of the entities, a warning message is shown and the categories are automatically created from the name of the entities. For example, a *C_tv_01* category would be created for entity *tv_01* in our model because we have not specified it.

4 Tooling for MLContext

The definition of a textual DSL involves the creation of the tools needed to support the language users. The basic tools are an editor to create DSL specifications, a parser to extract models from a DSL specification and a code generator which is able to transform the DSL models into software artifacts. Several tools currently exist which automatically generate editors and parsers from either the DSLs metamodel (e.g. TCS [JBK06]) or the DSLs grammar (e.g. Xtext [Xte10]). We have used TCS to apply a metamodel-based approach. With regard to the code generator, model transformations are written to obtain the desired artifacts. We have implemented a model-to-text transformation in the MOFScript language [MOF10] supported in the EMF framework of Eclipse. TCS is an Eclipse component of the GMT project that enables the specification of textual concrete syntaxes for DSLs by attaching syntactic information (e.g. keywords) to metamodel elements of the DSLs abstract syntax. TCS uses this specification to generate: i) an Eclipse editor, which features syntax highlighting, an outline and hyperlinks; ii) a parser (injector) in



(a) Partial view of the generated context model

```
public void run() \{
   int i = 0;
   while (i < NUM\_OPERACIONES) \{
      i++;
      try {
         Thread.sleep( (long)(WAIT_TIME) );
      } catch (InterruptedException ex) {
          i = NUM_OPERACIONES; continue;
        }
//---- Add adapter interface code here ----
   boolean returnValue;
   //returnValue=AdapterObject.isBurning();
   float r = (float)Math.random()*100.f;
   returnValue=r<50;
//---- End Adapter interface section ----
// check that context service is active
   waitForServiceActive();
// update context entity property
   cs.setContextItem( entityClass, entityId,
   "fire_presence", returnValue);
   }
// finalize
   ((OCPService)cs).terminate();
   System.out.println(id + " closed.");
}}
```

(b) Code example from the fire_alarm source producer

Figure 3: Generated artifacts

Java that takes a source model expressed in its textual concrete syntax and generates a model conforming to the DSL metamodel; and iii) an extractor that performs the reverse operation to the injector, and generates textual specifications from the models.

When generating software artifacts with MLContext., a model is extracted from a textual specification and transformed into a software artifact (using a model-text transformation).

Given the textual specification of the hospital example described in Section 3, the generated model is shown in Figure 3a in form of a tree model. This model conforms to the MLContext metamodel (see Figure 2). We have written a MOFScript model-to-text transformation to generate three OCP artifacts[1] : the ontology based on OWL-DL for the knowledge base of the system, the Java classes for producers and a Java skeleton of the main program, which initializes all context instances and producers. The individuals are created during run-time as instances of the OWL classes by the main Java program. The information needed for the Java code of the producers is obtained from the context information sources of the model. An example of the Java code generated is shown in Figure 3b.

5 Related work

Since the emergence of pervasive computing, great research efforts have been devoted to the subject of context information modeling and several approaches have been proposed. A survey

[1] All the generated artifacts are available at http://www.modelum.es/MLContext

of context model approaches is presented in [BDR07] and in [BCQ+07]. However, the number of proposals for model-driven development of context-aware and pervasive systems is considerably small, although it is expected to grow significantly in coming years. Most of these works include a solution for representing context information. The most relevant approaches will be analyzed below.

ContextUML [SB05] is a UML-based graphical language aimed at the design of context-aware web services, which include constructs for modeling context information. ContextUML distinguishes between atomic and composite context. However the notion of composite context is different from other approaches which also consider it. ContextUML defines a composite context as a state of a context depending on two or more atomic contexts, for example "($temperature > 40^o C$) OR ($rainLikehood > 80\%$)", and models it by using statecharts, while in MLContext a complex context is the aggregation of two or more simple contexts. Because ContextUML does not include the notion of entity, the contextual information is scattered through the model rather than being related to an entity as in MLContext. The sources of the contextual information are context services.

PervML [SVP08] is a DSL for specifying pervasive systems, which has been defined for a MOF metamodel and its concrete syntax is based on UML 2.0 notation. In contrast to MLContext, PervML is more oriented towards representing services than context and the context information is not separated from its sources. PervML uses a transformation engine to translate the PervML models to an OWL ontology.

An MDD approach for the development of context-aware systems is described in [ADB07], where a process with six phases is proposed. This process takes into account both the collection of the context and adaptation mechanisms, and the languages needed for each phase are defined by means of an UML profile. With regard to the context modeling, the identification of the required context information must be performed during the first phase, specifying several context elements such as the context types or the required context quality. These specifications must be made independently of the platform that will be used to collect the information. The UML profile for the context model includes the elements for collecting information (starting time of collection, number of samples, rate of sampling,...) which depends on a particular application.

To the best of our knowledge, all current approaches include target platform or application dependent aspects in the context model. Since MLContext proposes to represent this information in a separated model, the models are thus more readable and closer to the users conceptual domain. For instance, ContextUML does not allow the assignation of a different context source for the same attribute in each of the instances of it classes to be clearly expressed. In addition to this, the contextual information is not related to an entity and is scattered throughout the model. This makes it difficult to generate code from a ContextUML model to some middlewares like JCAF [Bar05], which makes use of entities. In JCAF, an entity is a small Java program that is executed in a context service. An MLContext model contains the information required to generate the JCAF code of the entities programs and part of the code of the context services.

It is also worth noting that ContextUML models can be expressed in MLContext with the exception of information specifying services, which must be specified in an implementation model. With regard to the artifacts generation, other approaches such us the Context Toolkit middleware [Dey00], do not make use of context representation and focus on services. In this case, the information of the MLContext model can be used only to generate a skeleton of the

widgets programmed in Context Toolkit.

6 Conclusions and future work

MLContext is the principal component of a generative architecture for context-aware applications currently under development. To the best of our knowledge, the approach proposed presents several innovative aspects such as providing a textual language tailored to context modeling and the generation of software artifacts from abstract models which do not include implementation or application details. We have shown a case study illustrating how software artifacts for a middleware platform can be generated from MLContext models. The MLContext tooling and examples can be downloaded and tested[2]. Since MLContext models do not include information related to the platforms or the specific implementation, these models can be reused in different context-aware applications which are based on the same context. In contrast to existing approaches, MLContext is entity-centered rather than category-centered, and it supports a taxonomy of types of context. These design choices have helped to provide a high level of abstraction which allows MLContext to be used by both developers and domain experts. MLContext is being used to generate OCP code in the CARDEA project[3] intended to create a platform of context-aware services for hospitals. From now on, our work will continue in two different directions. On the one hand, we are working on developing model transformations to automatically generate code for other context management middlewares for ubiquitous computing (e.g. JCAF, Context toolkit). On the other hand, there are practical aspects regarding context awareness that should be considered in the future. One of these aspects is the dynamics of context sources and data flows (i.e. sample rate, starter time of collection, number of samples, etc). The metamodel should be modified to include such details in future specifications. The other kinds of aspects include services provided by the middleware and directly related to context information (e.g. context discovery, reasoning), but also adaptive services which should be built on top of the corresponding middleware (e.g. OCP) in order to create complete ubiquitous computing applications on top of such middleware.

Bibliography

[ADB07] D. Ayed, D. Delanote, Y. Berbers. MDD Approach for the Development of context-Aware Applications. In Kokinov et al. (eds.), *CONTEXT 2007*. Volume 4635, pp. 15–28. LNAI, 2007.

[AFG+07] L. Ardissono, R. Furnari, A. Goy, G. Petrone, M. Segnan. Context-Aware Workflow Management. In *Int. Conf. on Web Engineering (ICWE 2007)*. Volume 4607, pp. 47–52. LNCS, 2007.

[Bar05] J. E. Bardram. The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for context-Aware Applications. In *Pervasive 2005*. Volume 3468. LNCS, 2005.

[2] http://www.modelum.es/MLContext

[3] CARDEA Project (TSI-020302-2008-78), Avanza Program, 2008 founded by Ministry of Industry, Tourism and Trade (Spain).

[BCQ+07] C. Bolchini, C. Curino, E. Quintarelli, F. A. Schreiber, L. Tanca. A Data-oriented Survey of Context Models. *SIGMOD Record* 36(4):19–26, December 2007.

[BDR07] M. Baldauf, S. Dustdar, F. Rosenberg. A survey on context-aware systems. *International Journal Ad Hoc and Ubiquitous Computing* 2(4):263–277, 2007.

[BVP+09] J. A. Botia, A. Villa, J. T. Palma, D. Perez, E. Iborra. Detecting domestic problems of elderly people: simple and unobstrusive sensors to generate the context of the attended. In *First International Workshop on Ambient Assisted Living. IWAAL, 2009*. LNCS 5602. 2009.

[DA00] A. K. Dey, G. D. Abowd. Towards a better understanding of context and context-awareness. In *Proceedings of the Workshop on the What, Who, where, when and How of Context-Awareness, New York, 2000*. ACM Press, 2000.

[Dey00] A. K. Dey. *Providing architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.

[EMF10] Eclipse Modeling Framework Project (EMF). http://www.eclipse.org/modeling/emf/, 2010.

[GM02] A. Göker, H. I. Myrhaug. User context and personalisation. In *6th European Conference on Case Based Reasoning, ECCBR 2002 ,Aberdeen, Scotland, UK, September 4-7, 2002, Workshop Proceedings* . Pp. 1–7. 2002.

[HIR02] K. Henricksen, J. Indulska, A. Rakotonirainy. Modeling Context Information in Pervasive Computing Systems. In *Pervasive 2002*. LNCS 2414, pp. 167–180. Springer-Verlang, 2002.

[HSKK09] J. Hong, E. Suh, J. Kim, S. Kim. Context-aware systems for proactive personalized service based on context history. *Expert Systems with Applications: An International Journal* 36(4):7448–7457, 2009.

[JBK06] F. Jounault, J. Bézivin, I. Kurtev. TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In *5th International Conference on Generative Programming and Component Engineering. Protland, Oregon, USA*. Pp. 249–254. 2006.

[KMK+03] P. Korpipää, J. Mantyjarvi, J. Kela, H. Keranen, E.-J. Malm. Managing context information in mobile devices. *IEEE Pervasive Computing* 2(3):42–51, July-September 2003.

[KP09] S. Kelly, R. Pohjonen. Worst Practices for Domain-Specific Modeling. *IEEE Software* 26(4):22–29, 2009.

[MHS05] M. Mernik, J. Heering, A. Sloane. When and how to develop domain specific languages. *ACM Computing Surveys* 37(4):316–344, december 2005.

[MOF10] MOFScript (Meta-Object Facility) model-to-text transformation tool. http://www.eclipse.org/gmt/mofscript, 2010.

[PB05] D. Preuveneers, Y. Berbers. Semantic and syntactic modeling of component-based services for context-aware pervasive systems using owl-s. In *MCMP-05. First International Workshop on Managing Context Information in Mobile and Pervasive Environments,2005*. Pp. 30–39. 2005.

[PMFS07] H. N. Pham, Q. H. Mahmoud, A. Ferworn, A. Sadeghian. Applying Model-Driven Development to Pervasive System Engineering. In *SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*. P. 7. IEEE Computer Society, 2007.

[PSM+05] L. F. Pires, M. v. Sinderen, E. Munthe-Kaas, S. Pokraev, M. Hutschemaekers, D. J. Plas. Techniques for describing and manipulating context information. Technical report, Freeband A-Muse Project, Deliverable D3.5, October 2005.

[SAW94] B. N. Schilit, N. L. Adams, R. Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*. Pp. 85–90. IEEE Computer Society, 1994.

[SB05] Q. Z. Sheng, B. Benatallah. ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services. In *2005 International Conference on Mobile Business (ICMB 2005), 11-13 July 2005, Sydney, Australia*. Pp. 206–212. IEEE Computer Society, 2005.

[Sch05] A. Schmidt. The Knowledge Maturing Process as a Unifying Concept for E-Learning and Knowledge Management. In *Proceedings of the 5th International Conference on Knowledge Management (I-KNOW 2005)*. 2005.

[Sel07] B. Selic. A Systematic Approach to Domain-Specific Language Desing Using UML. In *Tenth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2007)*. Pp. 2–9. IEEE Computer society, May 2007.

[SVP08] E. Serral, P. Valderas, V. Pelechano. A Model Driven Development Method for Developing Context-Aware Pervasive Systems. In *UIC 2008*. Volume 5061, pp. 662–676. LNCS, 2008.

[VK07] E. Vildjiounaite, S. Kallio. A Layered Approach to Context-Dependent User Modelling. In *Advances in Information Retrieval ECIR 2007*. Volume 4425, pp. 749–752. LNCS, 2007.

[Xte10] Xtext - a programming language framework. http://www.eclipse.org/Xtext/, 2010.