

A DSL for Context Quality Modeling in Context-Aware Applications

José R. Hoyos, Davy Preuveneers, Jesús J. García-Molina, and Yolande Berbers

Abstract Developing reliable context-aware applications remains a big challenge, even after a decade of research in this area. Usually a lot of code is required to handle an application's correct behavior in a variety of different situations. Along with a growing amount of code, also increases the risk of programming errors that may lead to an undesired behavior in particular situations. In this paper we present a domain specific language (DSL) for developing context-aware applications. It allows creating context quality models which are transformed into software artifacts of the final application. This code generation saves time and effort, and helps to ensure an appropriate autonomic behavior at runtime in inherently uncertain situations.

Key words: Domain Specific Language, Context-Aware, Quality, MDD, Context Quality

1 Introduction

Developing adaptive and proactive context-aware [2] systems is a complex process. It is a big challenge to achieve a reliable behavior and quality of service (QoS) in a variety of circumstances. Not only the implementation, but the whole design phase should unambiguously describe how the system should operate in different situations. Software engineering techniques like Model Driven Development (MDD) can

José R. Hoyos · Jesús J. García-Molina
Departamento de Informática y Sistemas
Universidad de Murcia, 30100 Murcia, Spain
e-mail: {jose.hoyos, jmolina}@um.es

Davy Preuveneers · Yolande Berbers
Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
e-mail: {davy.preuveneers, yolande.berbers}@cs.kuleuven.be

help to achieve this, using models at different levels of abstraction. These models can be expressed using well-defined languages called Domain Specific Languages (DSL) from which model transformations can generate code. This process saves time and effort to develop the system.

Pervasive computing is typically highly sensor-driven, but all too often so called smart context-aware systems wrongly assume that the information they retrieve from context sensors in the surrounding is accurate and relevant. Sensors are prone to noise and imperfections, and context information can be ambiguous. Having a notion of context quality helps to select reliable context sources, to be confident about a particular context situation, and to trigger appropriate proactive behavior.

The MDD approach has not been well exploited by the research community for context quality modeling. With most tools mainly focusing on the business logic of an application, it is very difficult to express specific context situations and context quality requirements to define and restrict an application's behavior.

MLContext is a DSL specially tailored to model context, which is part of an MDD framework we are developing for context-aware systems [4]. In this work we present an MLContext extension for context-quality modeling. With this extension, to the best of our knowledge, it is the first DSL which lets the developer not only model the context quality attributes but also the applications context quality requirements, keeping the context model details separated from the business logic details. Another contribution is the domain analysis we carried out to define the quality extension of MLContext. We have identified the most commonly used quality attributes in the literature. We will also show how they can be combined to compute context quality for aggregated contexts.

This paper is organized as follows. In section 2, a survey of current related work is presented. In section 3, the main issues related to the representation of context quality are analyzed in order to identify the elements for our DSL. In section 4, the DSL is presented by means of an example. We also explain how the context and quality models can be used for computing context quality. Section 5 shows automatic generation of code from these models. Finally, in section 6, we end this paper with conclusions and opportunities for future work.

2 Related work

A UML-based structured model of context quality is presented in [10]. This approach provides quality parameters for sensor and context situations. It also includes an aggregation model which describes aggregation of quality from sensor level to context situations. These models are subjected to UML notation restrictions and they can not specify in detail the context situations. Their aggregation model can only specify which quality parameters are related to a context situation. This approach does not generate any code from the models. MLContext does not use a separated aggregation model, as this kind of aggregation is implicit in our quality model. This can be achieved by defining the quality requirements for each context

Table 1 Quality attributes classification.

Data acquisition		Data representation		Data usage	
precision	accuracy	units	format	believability	relevance
location	timeliness	understandability	aliases	completeness	availability
coverage	range			comparability	

situation because they are related with the quality attributes of the sensors. Another approach for modeling context situations without quality requirements is described in [1]. The authors make a distinction between a model of the environment (which represents concepts that are relevant to the system and their relationships) and a model of context (which represents facts, made machine interpretable through this model). The first model is defined by using an ontology, and the second by using the Resource Description Framework (RDF). A context is represented in the form of a triple (e.g. $\langle \text{Adrian}, \text{hasLocation}, \text{Room01} \rangle$). This kind of representation has a low level of abstraction, and the authors report difficulties to model some advanced relationships for constraints such as "close to", which can be easily expressed using MLContext.

3 Domain analysis on context quality modeling

When defining a DSL, a domain analysis is required to identify the main concepts of the language and the relationships between them. We have analyzed several existing approaches [6, 7, 8, 9] on context quality with the aim of extending MLContext with constructs for modeling and measuring quality in a context. These approaches organize quality parameters in several dimensions, but they do not consider that quality issues at low level (i.e. sensor data) are different from those at higher levels of context (i.e. relevance and ambiguity) [5].

In some cases we found an overlap of quality attributes belonging to different dimensions, and even a different interpretation for the same quality attributes (e.g. accuracy and precision). We classified in Table 1 the most common quality attributes based on three categories: *data acquisition*, *data representation* and *data usage*. This classification is simple and avoids overlap. Our context model can be extended to include new quality attributes.

- **Data acquisition:** these are directly related to the sensors that captured the information and must be defined at the sensor level.
- **Data representation:** these are used to specify acceptable representations of information that can be correctly interpreted.
- **Data usage:** these define the context of the data acquisition itself to decide if the data is relevant.

As a result of this domain analysis, we have identified the following concepts: *entity*, *source of context*, *provider*, *situation*. An *entity* could be any person, place, physical or abstract object, or event relevant for the interaction between a user and

an application (cfr. Dey’s definition of context [3]). The value of the properties of an entity can be obtained from a *source of context*. This context source is attached to a physical adapter or *provider*. A *situation* is a set of constraints on contextual facts, i.e. *context information*, so that when these constraints are satisfied in the sensed environment, the situation is said to occur [1]. For example, the health status of a patient could be composed of his temperature, blood pressure and pulse with ‘*the patient is ill*’ being described with a constraint ‘*temperature > 37°C*’. The aim of this work is to compute the global quality of this context situation based on the quality of the pieces of context information it is composed of.

We have identified these requirements to extend MLContext:

1. The DSL must allow to express the following:
 - Quality attributes of context sources
 - Quality requirements for context-aware applications
 - Different context situations to support behavior variability in the design phase
 - Composition of complex context situations with aggregation of simple ones
2. The DSL must support inference of quality for aggregated contexts.

4 A DSL for context-quality aware applications

In this section we show how to model context quality using our DSL by means of a simple example. Let us suppose a user is in his car and he is trying to find an open gas station (e.g. in a 1 Km radius). There are four steps in our process of modeling: (1) Model entities, (2) Define sources of context, (3) Specify providers with quality attributes, and (4) Model context situations with quality requirements. We use the MLContext syntax for steps (1) and (2), and the MLContext quality extension presented in this paper for steps (3) and (4). The MLContext abstract syntax and concrete syntax is explained in [4].

Step 1: We will model the entities now. Our context model has three entities: the *user*, his *car* and the *gas_station*. The complex context of the *user* is composed of three simple contexts of *personal*, *environment* and *physical* type. The property “name” has a constant value “John” and is supplied by the user. The “location” property value can be obtained from the GPS of the *mobile_pda* source of context. The “time” property value can be obtained from the *mobile_pda* source but also from the system *clock*. The entity *car* has a simple context of environment type with a property “contains” expressing the fact that the *user* is in the *car*. The value of the “location” property of the *car* can be obtained from the *car_GPS* source of context. Finally, for the *gas_station* entity, we have modeled its “location”, the “opening” and “closing” times. We used the *quality* modifier to specify that the location value is expressed in WGS84 format. The accuracy of “0” expresses an exact value.

```

contextModel example1 {
entity user {
  personal { "name" : "John" }
  environment { "location" source: mobile_pda }
  physical { "time" source: mobile_pda, clock } }
entity car {
  environment { "contains" : user
    "location" source: car_GPS } }
entity gas_station {
  environment { "location" : "38.0237616129431, -1.17448434233665"
    quality: "units:WGS84, accuracy:0"}
  social { "opening_time" source: shops_system_service
    "closing_time" source: shops_system_service } } }

```

Step 2: Due to space constraints we only show the *mobile_pda* context source. The *interfaceID* keyword is a provider ID. The *methodName* statements are used to specify how the information is retrieved from the provider. "getLocation" and "getTime" are used to obtain the location of the user and the current time.

```

contextSource mobile_pda {
  interfaceID : "01-02-03-0A"
  methodName : "getLocation" {
    supply: user.location
    returnValue: "coordinates"}
  methodName : "getTime" {
    supply: user.time
    returnValue: "string"} }

```

The keyword *returnValue* specifies the type (format) of the returned value. The *car_GPS* and *clock* sources are defined in a similar way and are attached to providers "01-02-03-0B" and "01-02-03-0C" respectively.

We need another model for steps (3) and (4) to specify the quality attributes of the providers and the quality requirements for our application. This is an application dependent model, so following an MDD approach we define it separately from our context model.

Step 3: Each of the methods for supplying context information can have different quality attributes. Provider "01-02-03-0A" is linked to the *mobile_pda* context source. This provider can supply the location of the user (via GPS) and the current time, through the methods "getLocation" and "getTime". In this example, we have specified quality attributes for both methods. The location information is expressed in WGS84 units with a precision of 0.1E-12, but an accuracy of only 0.0064 (i.e. a 600m error approximately). The *timestamp* keyword indicates that this adapter can supply the time instant of the measurement. For the "getTime" method, we only specify that it provides the time using the SystemTimeDate units. Provider "01-02-03-0B" is attached to the *car_GPS* context source. It has the same precision as the *mobile_pda* but better accuracy (300m error approximately). The clock quality attributes are defined in a similar way but not shown here.

```

provider "01-02-03-0A" {
  location : user
  method : "getLocation" { units : "WGS84" precision : "0.1E-12"
    accuracy : "0.0064" timestamp }
  method : "getTime" { units : "SystemTimeDate" } }
provider "01-02-03-0B" {
  location : car
  method : "getLocation" { units : "WGS84" precision : "0.1E-12"
    accuracy : "0.0023" timestamp } }

```

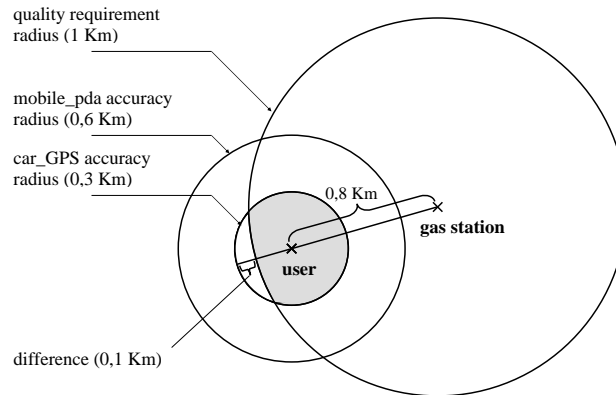


Fig. 1 Accuracy radius for the *gas_station*'s location.

Step 4: We can now express the quality requirements for our specific context-aware application. We are interested in finding a gas station near the user's location. Therefore, we can specify the following context situation:

```
situation open_gas_station_is_near {
  #user.location == #gas_station.location[accuracy="1", units="kilometers",
  relevance="1.2"]
  AND #user.time >= #gas_station.opening_time
  AND #user.time <= #gas_station.closing_time }
```

The *open_gas_station_is_near* situation is based on three context facts, which are specified by expressions concatenated by AND operators, so we compute the global quality for this situation as the product of the quality value of each of them. The first fact specifies that, if the user location and the gas station location are totally accurate, we accept as valid any reported locations which differ at most one kilometer. Of course, if the locations are not totally accurate, the quality of this context fact will not be 100%. *Relevance* is also an important subjective quality requirement. The developer must be able to express that certain context information is more important (relevant) than other ones and, therefore, its quality value has a higher contribution to the global quality of the context situation. In this example, we can suppose that it is more important to be sure about the distance from the gas station, because the user is interested in reaching the gas station (perhaps due to a low fuel level) even if he needs to wait until it is opened. The value of the *relevance* requirement is a factor which amplifies the maximum difference between the real value and the reported value of the location. By inspecting the models, the middleware knows it can obtain the value of the user's location property from the *mobile_pda* source. The user is in his car as may be inferred from the OWL-DL generated ontology, so the *car_GPS* can be added to the list of available sources for the user's location property. In this case, the middleware will choose the *car_GPS* source because it has better accuracy. Suppose the *car_GPS* reports a 0,8 Km distance from the *gas_station* (see Fig. 1).

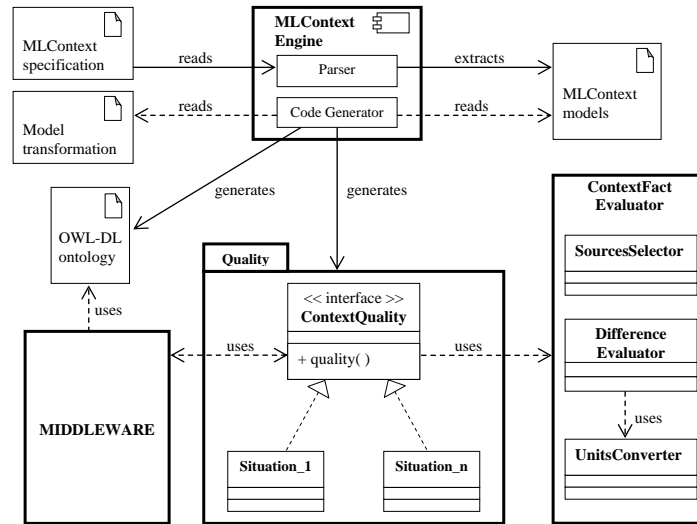


Fig. 2 Model-based framework to generate software artifacts for context quality.

Any user's location inside the 1 Km radius from the gas station will report a 100% of quality value but, there is a possibility that the user's actual position is outside the radius. The grayed zone of the accuracy circle for the *car_GPS* meets the requirements for the context situation we have defined. It represents a 83,33% of the total surface. Therefore, the quality value reported by the middleware for this fact might be 0,8333, but it is only 0,80 because of the relevance requirement.

The middleware can obtain the opening time for the *gas_station* from the *shops_system_service*. Let us suppose that this source provides the "10:00" value for the opening hour, with a 30 minutes accuracy. The current time (user.time) is "10:22", so there is an 8 minutes interval in which the *gas_station* might be closed. Therefore, the quality of this context fact is 86,67%. The quality value for the last context fact is 100% because current time is far from the closing time. The quality value reported for the global context situation will be $0,80 \times 0,8667 \times 1 = 0,6934$.

5 Automatic code generation

The MLContext engine automatically generates software artifacts of the context-aware application from a context model. An OWL-DL ontology and Java classes for context providers were generated for the OCP middleware as described in [4]. Whereas the generation of these Java classes was specific to OCP, the ontology could be used by other middleware. In this paper, the ontology is used by the middleware to infer that the user is in his car and, therefore, the car GPS can be used to obtain his position. An OWL *class* is created for each of the categories in the model

and a *DatatypeProperty* for each of their properties, taking into account the inheritance hierarchy. When a property refers to other entities (relationship), we create an OWL *ObjectProperty*. With regards to the MLContext quality model, a quality framework has been defined which can be used to provide functionality related to the quality management to any existing middleware. Fig. 2 shows a partial view of this framework and the generated artifacts by the MLContext engine. This engine automates the instantiation process by generating Java classes for each context situation. These classes implement the interface *ContextQuality* which has a quality method for computing the quality value of the situations based on the quality of their context facts. The evaluation of the quality of a context fact is performed by a *ContextFact Evaluator*. It uses a *Difference Evaluator* which calculates the maximum difference between the required value of a context property and the expected value from the context source. The framework also includes a *UnitsConverter*, which can change from some units to others, and a *SourceSelector* for selecting the (most suitable) best context source to obtain the value of a context property if it has more than one source. Fig. 3 shows an excerpt from the generated Java code for the example presented in this paper.

```
public class open gas station is near implements IContextQuality {
    private float CF1; //user.location == gas station.location
    private float CF2; //user.time >= gas station.opening time
    private float CF3; //user.time <= gas station.closing time

    public open gas station is_near () {
        ContextFactEvaluator CFE = new ContextFactEvaluator ();
        List<QualityReq> LOPQR1 = new ArrayList<QualityReq> ();
        LOPQR1.add(new QualityReq("accuracy", QualityToolsPackage.OPEQUAL, "1"));
        LOPQR1.add(new QualityReq("units", QualityToolsPackage.OPEQUAL, "kilometers"));
        LOPQR1.add(new QualityReq("relevance", QualityToolsPackage.OPEQUAL, "1.2"));
        CF1=CFE.evaluate("user.location",
            "gas_station.location", QualityToolsPackage.OPEQUAL, LOPQR1);
        CF2=CFE.evaluate("user.time",
            "gas_station.opening time", QualityToolsPackage.OPGREATER, null);
        CF3=CFE.evaluate("user.time",
            "gas_station.closing time", QualityToolsPackage.OPLESSE, null);
    }
    public float quality () {
        float QualityValue;
        QualityValue=CF1*CF2*CF3;
        return QualityValue;
    }
}
```

Fig. 3 An excerpt from the generated Java code.

6 Conclusions and future work

In this work, we have presented a DSL specially tailored and designed for context quality modeling following a MDD approach. It is platform independent, and the application dependent aspects are defined in a separated model from the context aspects, so the context model can be reused in different applications. The DSL allows the developer to express not only objective quality attributes (like precision of a source of context) but also subjective ones (like relevance of the context infor-

mation). We have developed an editor, with syntax highlight and code completion. We have also written some MOFScript transformations to automatically generate an OWL-DL ontology and Java code from the context model. Another contribution is the domain analysis we have performed in the designing of our DSL. We have identified the most used quality attributes in the literature and their existing relationships. As a future work we are planning to extend the code generation to different middlewares and to use the context model for predicting context situations.

References

1. Clear, Adrian K. and Dobson, Simon and Nixon, Paddy. An approach to dealing with uncertainty in context-aware pervasive systems. In *UK/IE IEEE SMC Cybernetic Systems Conference 2007*, Dublin, IE, 2007. IEEE Press.
2. Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, February 2001.
3. Anind K. Dey and 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.
4. Hoyos, José Ramón and García-Molina, Jesús and Botia, Juan Antonio. MLContext: A Context-Modeling Language for Context-Aware Systems. In *3rd DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services 2010*, volume 28, 2010.
5. McKeever, Susan and Ye, Juan and Coyle, Lorcan and Dobson, Simon. A multilayered uncertainty model for context aware systems. In *proceedings of the International Conference on Pervasive Computing: Late Breaking Result*, pages 1–4, May 2008.
6. Diane M. Strong, Yang W. Lee, and Richard Y. Wang. Data quality in context. *Commun. ACM*, 40(5):103–110, 1997.
7. Yair Wand and Richard Y. Wang. Anchoring data quality dimensions in ontological foundations. *Commun. ACM*, 39(11):86–95, 1996.
8. Wang, Richard Y. and Strong, Diane M. Beyond accuracy: what data quality means to data consumers. *J. Manage. Inf. Syst.*, 12(4):5–33, 1996.
9. Stephanie Watts, G. Shankaranarayanan, and Adir Even. Data quality assessment in context: A cognitive perspective. *Decis. Support Syst.*, 48(1):202–211, 2009.
10. Juan Ye, Susan McKeever, Lorcan Coyle, Steve Neely, and Simon Dobson. Resolving uncertainty in context integration and abstraction: context integration and abstraction. In *ICPS '08: Proceedings of the 5th international conference on Pervasive services*, pages 131–140, New York, NY, USA, 2008. ACM.