Context-aware Model Driven Development by Parameterized Transformation

Samyr Vale[1,2], Slimane Hammoudi[1]

[1] ESEO, 4 rue Merlet de la Boulaye BP 926, 49009 Angers cedex 01 France
{samyr.vale, slimane.hammoudi}@eseo.fr

[2] LERIA, Université d'Angers, 2 boulevard Lavoisier, 49045 Angers cedex 01 France

Abstract. Context-aware development has been an emergent subject of many research works in ubiquitous computing. Few of them propose Model Driven Development as a standard on context-aware development. Many focus on context capture and adaptation by the use of legacy architectures and others artifacts to input context into application logic. This work proposes Model Driven Development to promote reuse, adaptability and interoperability in context-aware applications development. By concerns separation in individual models and by transformation techniques context can be provided, modeled and adapted independently of business logic and platform details. We also present in this paper our context metamodel proposition based on ontology concepts and the parameterized transformation technique applied to context-aware development.

Keywords: Ubiquitous computing, MDA, Parameterization, Context-awareness

1 Introduction

Nowadays, distributed B2B and B2C applications aggregate more functionalities and implement more complex business logic. Empowered by internet resources, enterprises increase application integration with their partners to provide better and more services for internet users.

The new tendency in software development targets mobile devices as client applications. Mobile phones and others handhelds are being used to access distributed applications everywhere and at any moment. Meteorological information, shopping and GPS guides are some examples of applications accessed by these device types.

Mobile web-supported devices simplify ubiquitous applications development. The term ubiquitous was firstly used by Weiser [1]. He idealized a physical environment with computational devices integrated (sensors, for example). The goal was to help users in their daily activities, in a transparent way, by the use of a computational environment with non-traditional devices. Nowadays, ubiquitous applications development has been studied by various research groups. Abowd *et al.* [2][3]

presented four issues in ubiquitous computing: *natural interfaces*, *capture and access of human activities*, *context-aware applications* and *everyday computing*.

We focus on context-aware applications. That means applications which use context to realize required activities and to provide relevant information for users without human interaction. The majority of context-aware applications implement business and contextual activities and data together. This programming practice hampers software reuse and context management.

Motivated by this problem we apply MDD™ (Model Driven Development) in context-aware applications development. OMG™ (Object Management Group) recommends the MDA® (Model Driven Architecture) [4] approach as a standard[1] in Model Driven Development.

Models are the best way to represent the structure and the semantic of the concepts manipulated by a system. By the separation of concerns (business and context) we improve reuse, interoperability, adaptability and management of context information. Context models can be built as independent pieces of application models and at different abstraction levels then attached by transformation techniques. We have investigated different techniques to make context adaptation and we propose some solutions for context integration into application model.

We present in this work our context metamodel focusing on the context types most used in ubiquitous development and our parameterized transformation proposition. We also have made some extensions in the OCL language to adapt some model transformation operations used to attach context into application models.

This paper is organized as follows. Section 2 presents the state of the art in context-aware applications development and gives a brief discussion of related works. Section 3 presents and describes our context metamodel. In Section 4 we expose the application of Model Driven Architecture in context-aware development and we also describe the use of the parameterized transformation for context adaptation. Finally we present our conclusions and ongoing activities about this subject.

2 Context-aware Applications

Context-aware application is a topic of the Ubiquitous Computing area. In most cases, applications in this domain use some information that characterizes a particularity of the user or the system itself to be more adaptable and to respond to user needs. *User name*, *location*, *time*, *device type* and *profile* are the most used contextual information.

Dey [5] expanded context notion and defined it as *"any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves"*.

This definition has been largely adopted by the academic community.

[1] *Model Driven Development (MDD), Model Driven Architecture (MDA), OMG are Trademarks of Object Management Group, Inc.*

Nevertheless, this diversity of context meaning leads context-aware programming more complex.

Some challenges found in context-aware applications development are:

- ✓ Context capture,
- ✓ Context representation,
- ✓ Context interpretation,
- ✓ Context adaptation,
- ✓ Context management,
- ✓ Context reuse.

To illustrate a common problem in context-aware application development, imagine a system whose interface is developed to be displayed in a specific handheld screen. If another device tries to display the same application's interface, it may not be displayed on account of its different screen size.

To make this interface adaptation, system developers have to change the interface parameters into the application code. This simple activity requires modifications in many phases of the application development process and code recompilation.

Some recent works in context-aware application development separate context information from application code.

Sheng *et al.*[6] propose an UML based language for the development of context-aware web services. In their language metamodel a context class is responsible for modelling context information. They classify context information on composite or atomic. Composite context can provide multiple context vocabularies. Like others works in the same domain, a particular mechanism to link context with web services is required.

Ou *et al.* [7] have been applied MDA in context-aware application development. They focus on the development of context-awareness based on ontologies. However, they did not explore the use of transformation techniques.

Ceri *et al.* [8] propose a model language for the development of context-aware web applications. They focus on context adaptation actions, context data representation and management.

Different context formalisms and representation types have been used as XML [9], ontologies, data bases, agents and others. But usually context adaptation depends on particulars infrastructures like *interpreters*, *wrappers*, *parsers*, *aggregators*, *adaptors* and other legacy artifacts.

All this arduous work developing context-aware application leads systems designer and programmers to insert context into application code.

3 Context Metamodel

The OMG's MDA aims to provide a set of standards for Model Driven Development. The principal concept in Model Driven Development is to standardize software development by an approach based on models. MDA focus on business and technical concerns separation in individual models by the concepts of PIM (Platform Independent Model) and PSM (Platform Specific Model). It also introduces transformation language concepts, transformation rules and transformation engine.

In MDA approach everything is model. Each model is based on a metamodel and a metamodel is based on a meta-metamodel. The MDA stack is separated in four different abstraction levels: the meta-metamodel (M3) level (the most abstract one), the metamodel (M2) level, the model (M1) level and the application execution level (M0). Although, M0 does not correspond to a model it is represented as one.

A meta-metamodel is a language model for expressing a metamodel (e.g. MOF). A metamodel is a language model for expressing a model (e.g. UML, EDOC)[10].

Some authors identified that model representation affords considerable benefits to context-aware development and context representation. Henricksen and Indulska [11] defined a context graphical language to model context data.

Tao Gu *et al*. [12] propose a context model based on OWL ontologies. The principal reason of their choice is the capability of supporting semantic interoperability to exchange, share and understand context information. They worked with a set of different contexts provided by sensors, agents and others. So they separate context in different domains to facilitate processing and interpretation by mobile clients.

By the diversity of meaning, to develop a context meta-model expressing a generic context is a difficult task.

We suggest the context representation using ontology concepts. "*An ontology defines the common terms and concepts (meaning) used to describe and represent an area of knowledge*" [13].

Ontologies have been used to represent complex knowledge and to improve semantic and consistence to web applications. As aforementioned, there are some works using ontologies to represent context information due to their variety of meanings and sources.

Fig.1 presents our context metamodel based on ontology concepts and represented by an agnostic UML formalism and based on ODM (Ontology Definition Metamodel) [13] principles. ODM is the OMG MDA standard for ontology development.

The context metamodel represents the context domain and definitions. We use W3C's RDF (Resource Description Framework) to represent context information at model (M1) level. RDF is used to describe and represent metadata associated with resources. RDF is the formalism used in the ODM metadata layer. So it fits to context representation provided in most cases by sensors or others artifacts but also by URI (Uniform Resource Identifier) references. The context domain and data can be described for developers and referenced as a URI.

RDF Schema (RDFS) is the abstraction of RDF and is MOF supported. RDFS permits definitions of vocabularies to represent context in metamodel level.

RdfsContextElement, as described in Fig.1, is a RDF Resource that represents context elements. Resource is a semantic widespread concept to represent context in metamodel level [14].

ContextElement represents any element that denotes context.

RDFSContainer represents URI context references or external context sources.

RDFSContextProperty represents context elements attributes, associations and values.

The advantages compared with UML attributes and associations are the independence provided by Properties. Property can be defined independent of associations as the opposite of UML classes.

RDFSContextDataType groups context data. We are interested in *profile*, *device*, *location* and *time* context information due to their importance in most of context-aware applications, as represented in the model (M1) level in Fig.1.

The data type is compatible with XML Schema abstraction and consists of the triple: *lexical space*, *value space* and *lexical to value mapping*. A data type can also be identified by a URI reference [14].

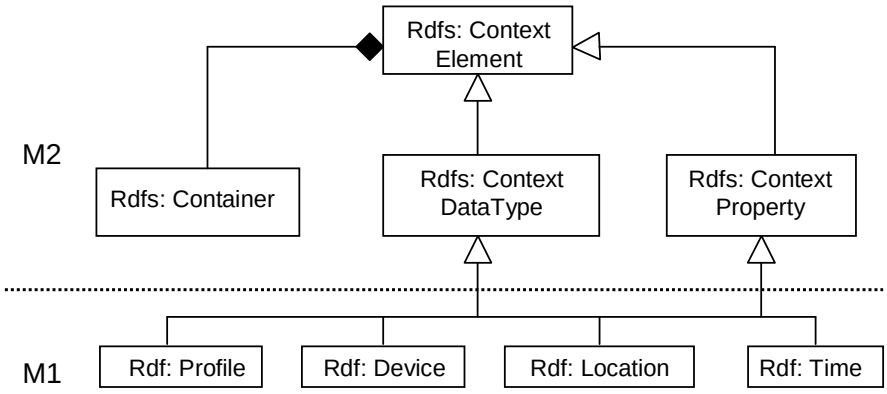

Fig. 1. Context Metamodel

As above mentioned, context reasoning is a challenge for developers. In some cases the types of context information (Profile, Device, Location and Time) can change in different applications. In certain ones, location for example, is not contextual information whilst in others it is.

Applications can not adapt themselves to any context information due to its variability of meanings. So specification of context scope is necessary to provide context reasoning for applications.

The RDF of our context model can be represented by XML as follows:

```
<!-- example -->
<rdf:RDF xmlns:rdf=«http://www.w3.org/
1999/02/22-rdf/syntax-ns#»
xlmns:ex=«http://profile.org/schema/»>
<Profile rdf:ID=«name»>
<Device rdf:type=«MobilePhoneModel_M»/>
<Location rdf:position=«x,y»/>
<Time rdf:hourminute=«hh:mm»/>
</Profile>
</rdf:RDF>
```

4 Context-awareness and Model Driven Architecture

The UML is the most used language to model oriented object systems and OMG's MDA has enhanced the UML benefits by separating PIM (Platform Independent Models) from PSM (Platform Specific Models) and by the use of transformation techniques between these models. This approach using concerns separation leads developers to decompose tasks and to represent them in different models.

The most important concepts in MDA are the mapping and transformation techniques. UML PIM models can be mapped to XML, OWL, EDOC, BPEL and others formalisms and a PIM model can be transformed to different platforms (PSMs) like Web Services, CORBA or EJB.

Transformations provide to Model Driven Development a large flexibility in reuse, adaptation and interoperability. Some techniques and different operations have been defined, evolved and applied in models transformations.

There is no consensus about transformation terminologies nor transformation patterns. MOF QVT [15] is the OMG proposition of a transformation language and it presents two principal models operations: mapping and transformation.

Mapping is the "*specification of a mechanism for transforming the elements of model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel"* [15].

For many authors mapping and transformation operations mean the same concept. In [16], the authors discuss the importance of distinguishing and separating the concepts of mapping and transformation in the process of transformation in MDA. In our context, we have analyzed some different techniques in the development of model driven context-aware applications and propose the best fitted solution for context-aware model driven development as described in the next section.

4.1 Parameterized Transformation

From PIM to PSM transformations are the heart of the MDA approach. The advantage of different models construction, as PIM and PSM is the treatment of different concerns in individual models. Different target platforms and languages can be used to the same business logical represented by a PIM model.

The transformations can also be realized between models of the same type, i.e., a PIM model can be modified by transformation techniques to another PIM model. The same can occur with PSM models. This type of transformation is mainly used to refine models.

Parameterized transformation consists on any model transformation based on parameters. This transformation technique is not explored and there is not a standard transformation language implementing it.

"A parameter specifies how arguments are passed into or out of an invocation of a behavioural feature like an operation. The type and multiplicity of a parameter restrict what values can be passed, how many, and whether the values are ordered" [10].

Frankel [17] mentions the importance of parameterization in model operations using the association of tagged values with PIM and PSM models. Tagging model elements allows the model language to easily filter out some specific elements.

Transformation by parameter could be used to improve new functionalities (values, properties, operations) or to change the application behaviour (activities).

In this work we present parameterized transformation focusing on PIM to PIM transformations.

The designer must specify the parameters to be inserted at the transformation phase. In our proposition these parameters are context or context-aware and after the transformation the application will join the context information specified into the parameters as illustrated in Fig.2.

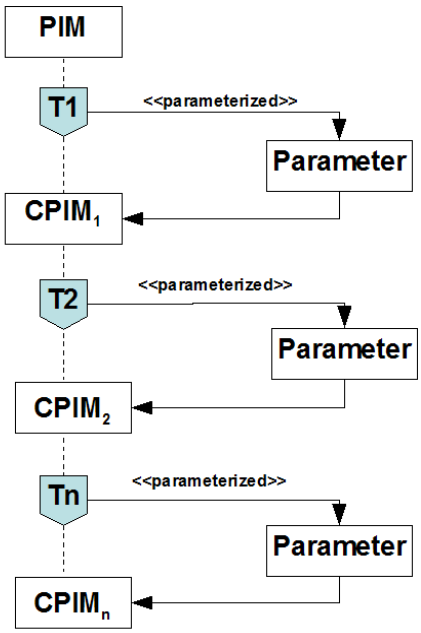

Fig. 2. Parameterized Transformation Concepts

A PIM model can be developed without contextual details. User name, profiles, device type, location can be added as parameters in transformations. The same PIM can be re-transformed and refined many times adding, deleting or updating context information.

We named PIM with context as CPIM (Contextual PIM). We can have different CPIMs of the same PIM, i.e., the same business logic can be adapted to different contexts by the contextual parameters adaptation. The CPIM fits together business requirements with contextual activities, properties and data. A CPIM can generate a CPSM (Contextual PSM) by the traditional transformation techniques, as shown in Fig.3. The CPSM inherits business requirements and context from the CPIM.

CPSM specifies operation system requirements, programming languages, middleware architectures and networking.

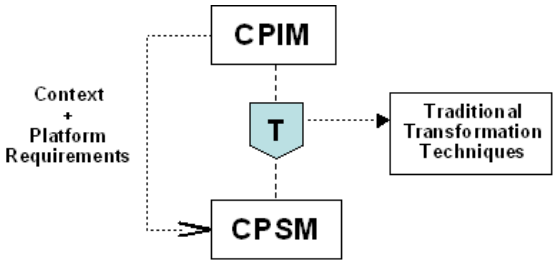

Fig. 3. From CPIM to CPSM Transformation

The designer has to specify into the application model the elements that will receive the context information. A mark, identified by the symbol #, is given for these elements to be recognized by the transformation engine. The marked elements represent context-aware elements, in others words, the model elements that can be contextualized.

The transformation language must be parameterized supported. In our case the parameters can be a Context Property and/or a Context Data Type.

We use templates to specify which elements in application model are potentially context-aware as depicted in Fig. 4.

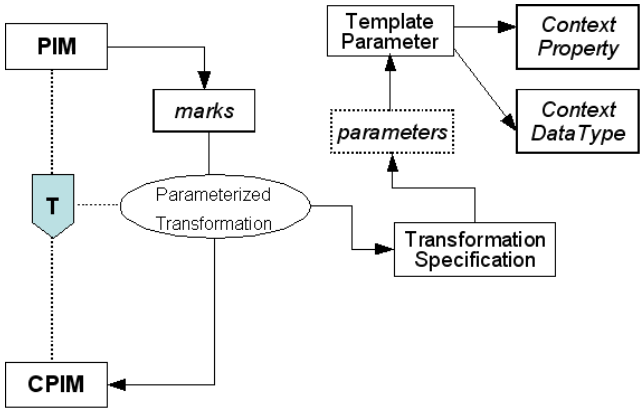

Fig. 4. Context Adaptation

The transformation engine has to navigate by the PIM model verifying the parameters and the elements marked and then make the transformation.

Template parameter [10] is an element used to specify how classifiers, packages and operations can be parameterized. UML 2.0 presents that any model element can be *templateable*.

For independent context-aware models we need to identify context elements that could be *parameterable*. A *parameterable* element is an element that can be exposed

as a formal template parameter for a template, or specified as an actual parameter in a binding of template [10].

Context parameter can be expressed as constraint and compared with the elements signature in template parameter. This operation is named as *match* operation. UML presents a Template Signature element that defines the signature for binding the template.

We resume our device adaptation example for a particular user. In Fig. 5 the PIM model identifies a user that uses a device. The designer is not pressed on define the type of the device in the PIM model. The context-aware elements, the device type in this example, are marked with the # symbol to indicate to the transformation engine the elements that will be contextualized. The transformation consists on context information adaptation.

The User Element, presented in Fig. 5, has a Device Type parameterable property expressed as a constraint in the template parameter. The *match* operation compares the parameter expressions.

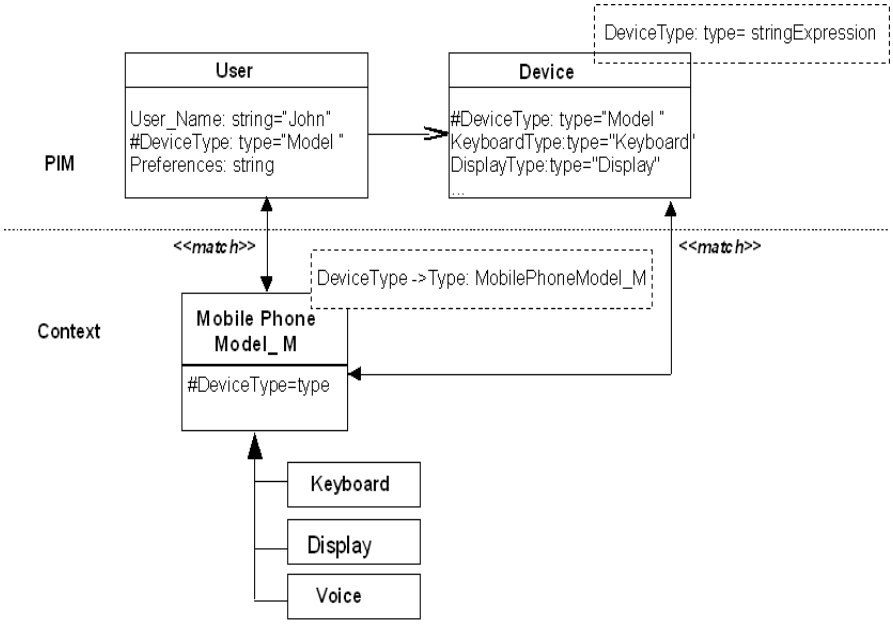

Fig. 5. Parameterized Transformation

The context of the *MobilePhoneModel_M* is attached after the comparison.

After the parameterized transformation the *DeviceType* property of *User* will be connected with a particular *Device* (context) element to supply to this model the characteristics of the mentioned device. This is made by the *includes* operation at instantiation time. In Fig. 6 we present our parameterized transformation metamodel.

Differently from traditional model transformations, the parameterized one has as source model a set of contextual parameters and as target model the PIM marked

model. The designer determines which model element will be transformed by tagging parameters.

The *match* operation is realized before the transformation one. The match binds and checks the concerned (marked) elements that will be transformed. It is also responsible for determining if a parameterable element is compatible with another one.

Semantical interpretation among these elements can be supplied by the ontologies use. Our context metamodel is ontology supported by the RDF interpretation. According to W3C RDF Semantic [14] a RDF logical semantic is identified by a *triple(x,y,z)* where *x* and *z* are semantical elements, data types or resources in our case (represented by a string), and *y* is the relation between them. The context can be represented by N-triples in URI references.

Nevertheless, we propose a semantic solution for context representation; this paper does not focus on context interpretation.

In Fig. 7 we illustrate some examples of contextual RDF triples for our scope of contextual elements. As aforementioned a rdf:property (instance of rdfs:property) is MOF supported and can be a UML relationship, attribute or value.

The Match class, as illustrated in Fig. 6, is also responsible for navigating over models. The OCL Rules class specifies the navigation rules using OCL (Object Constraint Language).

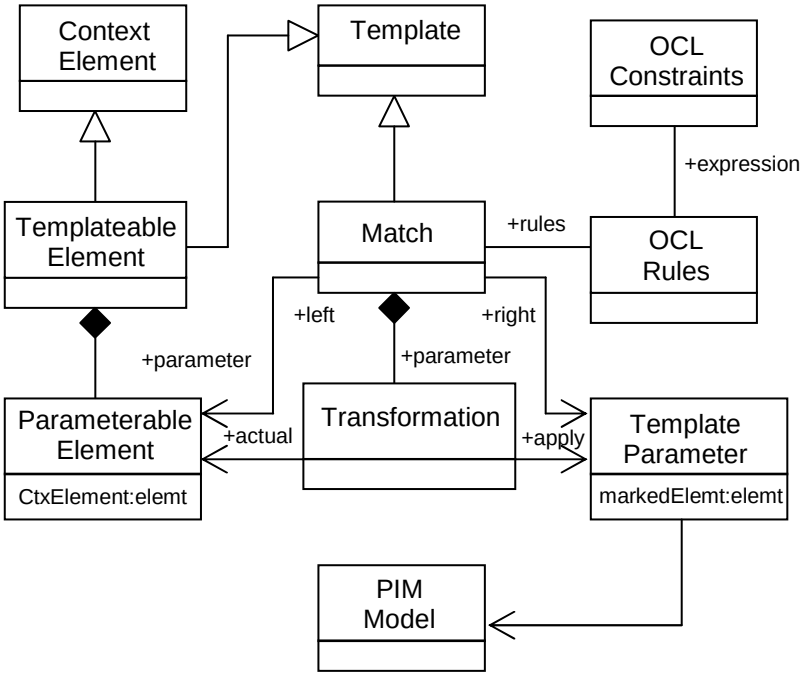

Fig. 6. Parameterized Transformation Metamodel

OCL permits filter expressions to add platform requirements and context information. The *match* operation generates the correspondences between the elements of the Parameterable Element and its correspondents into the Template Parameter. This can be realized by the use of the new SQL queries supported by OCL 2.0.

OCL owns a set of types and operations defined in its OCL Library. Some of the types are integer, string, real and boolean. Although, OCL is easily adapted for new types insertion and provides mechanisms for language extension. For example, the *let* expression permits definitions of variables and expressions. OCL also allows attachment of the new variables defined to a method or property.

```
rdf:profile, rdf:property, rdf:profile
rdf:location, rdf:property, rdf:location
rdf:device, rdf:property, rdf:device
rdf:time, rdf:property, rdf:time
rdf:profile, rdf:property, rdf:time
rdf:device, rdf:property, rdf:location
rdf:location, rdf:property, rdf:time
...
```

Fig. 7. Contextual RDF Triples

We define some OCL [18] extensions with the presence of the *match* operation. As aforementioned, the match operation checks the correspondence of the elements evolved in the parameterized transformation. The return value can be a type, property or N-triple.

The match navigates over the model searching the marked elements and their correspondences. The left and right models elements must have the same signature to be interpreted as correspondents.

In other words, the constraint rules of the *match* operation identify that *User_ID*, *UserId* or *UserID* has the same meaning. The semantic meaning of context elements is expressed in the ontology vocabulary domain URI referenced.

The *match* operation products the correspondences and the transformation operation applies context represented by the parameterable Element into the Application Template Parameter. The match and transformation operations are partially present as follows. The complete match rules for context-aware development will be presented in future works.

```
match (TemplateParameter.markedElemt.
allparameters ->
collect(CtxElement |CtxElement.
ParameterableElement.
type(x,y,z) and
collect(MarkedElemt |markedElemt.
TemplateParameter.
type(x,y,z) and
select->(CtxElement |CtxElement.
ParameterableElement.
type(x,y,z)isCompatiblewith
MarkedElemt |markedElemt.
TemplateParameter.
type(x,y,z)
result= select match{
correspondences->collect (c|match.
correspondences.at(index))}
Self -> transform (TemplateParameter |
TemplateParameter.Type(x,y,z).markedElement->
includes(parameterableElement))
```

5 Conclusion

In this work we presented our approach for context-aware application development in the context of Model Driven Architecture. Everything is considered as model in MDA approach and the application development is realized by a set of different abstraction levels construction and the use of different transformation techniques.

We identify some important benefits of MDA as concerns separation, reuse of models and interoperability. These features are skipped in many context-aware applications.

By the application of MDA approach we supply its benefits in ubiquitous computing. OMG's MDA proposes separation of platform details in different models. We separate context information from business logic in individual ones.

We also propose the use of template parameters and the identification by a specific mark of the context-aware elements in PIM models.

By the use of a particular transformation technique the contextual parameter identified into the model will be contextualized with the parameterable element which represents context information.

To accomplish this we propose parameterized transformation implemented with OCL language. OCL is a pattern language commonly used for models operations. It has many advantages as traceability concerns, UML supported and adaptability.

We propose our parameterized transformation metamodel composed principally of *match* and *transformation* operations. Some extensions in OCL have been realized to provide these functionalities.

We also propose for context information definition ontologies concepts represented by RDF and RDFSchemas. Contrary to UML model, ontologies

represented by RDF models supply reasoning and independence in context information and are MOF supported.

As a future activity about this work we are implementing our parameterized transformation engine based on the OCL language. The extensions proposed in OCL syntax will also be implemented.

References

1. Weiser, M. The Computer for the 21s century, Scientific American, 94-104 (1991)
2. Abowd, G.D., Mynatt, E.D., Charting Past, Present and Future Research in Ubiquitous Computing. ACM Transactions on Computer-Human Interaction, 7(1).29-58(2000)
3. Abowd, G.D., Mynatt, E.D. Rodden. T, The Human Experience. IEEE Pervasive Computing.(1), 48-57(2002)
4. OMG(Object Management Group).Model Driven Architecture (MDA), document number ormsc/2001-07-01(2001)
5. Dey A.K. , Salber D. and Abowd G. D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications., Human-Computer Interaction Journal, pp. 97–166(2001)
6. Sheng Q. Z., and Benatallah B., ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services, The 4th International Conference on Mobile Business (ICMB'05), IEEE Computer Society. Sydney, Australia(2005)
7. Ou S., Georgalas N., Azmoodeh M., Yang K. and Sun X., A Model Driven Integration Architecture for Ontology-Based Context Modelling and Context-Aware Application Development, A. Rensink and J. Warmer (Eds.): ECMDA-FA 2006, LNCS 4066, pp.188–197. Springer-Verlag Berlin Heidelberg (2006)
8. Ceri S., Daniel F., Matera M., Facca F.M., Model Driven Development of Context-aware Web Applications, ACM Transactions on Internet Technology(TOIT), ACM Publisher, Volume 7, Issue 1, (2007)
9. W3C. XML specification. W3C document REC-XML11-200400204
10. OMG. UML 2.0 Superstructure, OMG document ptc/03-08-02, 2003.
11. Henricksen K, Indulska J., and Rakotonirainy A., Modeling Context Information in Pervasive Computing Systems, F. Mattern and M. Naghshineh (Eds.): Pervasive 2002, LNCS 2414, pp. 167–180, 2002.Springer-Verlag Berlin Heidelberg (2002)
12. Tao Gu, Xiaohang Wang, Hung Keng Pung, Daqing Zhang: An OWL-based Context Model in Intelligent Environments. Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS'04), San Diego, California(2004)
13. OMG. Ontology Definition Metamodel. OMG document OMG/ RFP ad/2006-05-01(2006)
14. W3C.RDF Semantics, W3C document rdf-mt, 10-02-04.
15. OMG. QVT-Merge Group. Query, View and Transformations for MOF 2.0. OMG document RFP (ad/2002-04-10)(2004)
16. Lopes D., Hammoudi S., Bézivin J., Jouault F., Generating Transformation Definition from Mapping Specification: Application to Web Service Platform, The 17th Conference on Advanced Information Systems Engineering (Caise'05) LNCS 3520, 309-325, Porto-Portugal (2005)
17. Frankel S. David., Model Driven Architecture: Applying MDA to Enterprise Computing, Wiley Publishing, Inc(2003)
18. OMG. UML 2.0 OCL Specification, OMG document ptc/03-08-08(2003)