CMP: A UML Context Modeling Profile for Mobile Distributed Systems

Christof Simons
Distributed Systems Group
Otto-Friedrich-Universität Bamberg
96045 Bamberg, Germany
christof.simons@wiai.uni-bamberg.de

Abstract

This paper proposes the Context Modeling Profile (CMP), a lightweight UML extension, as a visual language for context models in mobile distributed systems. The resulting models visualize meta information of the context, i.e. source and validity of context information, and reflect privacy restrictions. The profile provides several well-formedness rules for context models supporting the development of context-aware mobile applications through an adequate visual modeling language. A case study is used to illustrate the approach.

1 Introduction

The context of a user is an interesting research topic in mobile computing. The user context enables technologies like context-based provision, discovery and usage of services and context-aware applications [2]. These technologies seem to be a promising way to enhance the capabilities of mobile devices, making them appear more intelligent, eliminate interface restrictions and leverage the support of the user by the device.

The context of the user can be defined as the set of information about the user himself and his environment [10]. Typical information in the context of a user are his name, his location, his current activity or persons nearby [1]. Due to the mobility of the user and his device, the context of the user is not a steady set of information but is changing irregularly.

In the development process of a context-aware application that should adapt to the user context the developer has to define, which context information are relevant for the adaption of the application. The structure of the context, i.e. the properties and the connections between the information have to be provided. Hence, the developer has to build an application specific model of the user context, resulting in an artifact of the development process, the context model.

The context model must not only contain the information type definitions but also reflect meta information about the context. This means for instance that it must indicate that the birth date of a person never alters but the current location of a person changes frequently. This is essential when the application should base upon a framework for context-aware applications. The services which are provided by the framework like storage support or checking the validity of the context highly depend on these meta information.

Visual Languages play an important role in software engineering because graphical models are better readable and understandable by human beings. The most popular object oriented modeling language is the UML (Unified Modeling Language, [7]). The UML provides different diagram types and also allows the separation of concerns by using several diagrams to focus different aspects of a software system. A huge number of UML based development environments exist, offering features like model transformation, model validation and code generation.

This paper shows how the context of a context-aware application for mobile distributed systems can be modeled using specialized UML class diagrams. Therefore, the general characteristics of the context are explained and the context model of a meeting system is proposed in section 2. The shortcomings of this approach are discussed afterwards and the usage of a UML profile is proposed as a solution. In section 3 CMP, our Context Modeling Profile, is introduced. In section 4 the context model of the meeting system is improved by applying CMP to the original context model. Related work is referred to in section 5 and the final section provides a conclusion and future work.

Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)
0-7695-2755-8/07 $20.00 © 2007 IEEE
1530-1605/07 $20.00 © 2007 IEEE

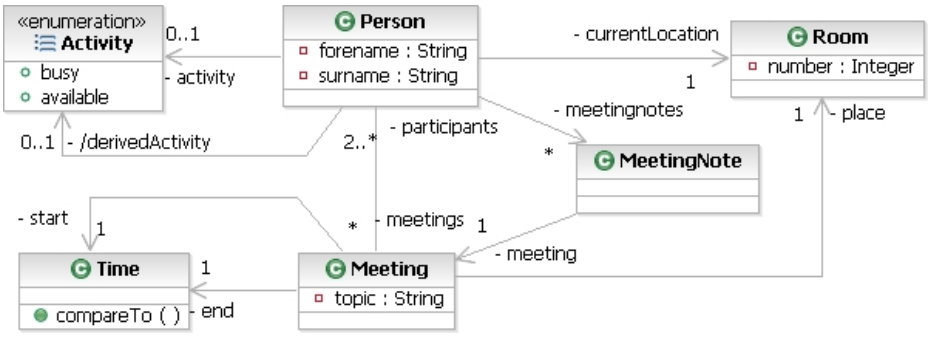

Figure 1. Extract of the context model of the meeting system

2 Case Study: Meeting System

The meeting system provided as a case study should support members of a university department in making appointments. This means that the schedules of several department members have to be matched to find a free slot for a meeting. Information about the user's current activity and location should be provided by the system, making it possible for other persons to query these information. When attending a meeting the activity of the user should be set by the system to *busy* automatically, indicating that the person should not be interrupted by other users of the system. The participants of a meeting should also be able to take personal notes during the meeting. These notes should be associated with the meeting automatically such that the user can access these notes more easy later on.

2.1 Technical requirements

The meeting system should support users in a mobile distributed system which are using mobile devices like PDAs. One important requirement is that the system must not depend on a central component like a storage node for storing the context of the users. The context of the user can contain personal information, hence privacy issues have to be regarded. Privacy is basically a problem of control [15]. In order to leverage the user's control over his personal information a central storage node should not be used. Therefore, the *global context*, the context information of all users, has to be distributed among the users of the system.

The mobile device of a user should only store the *individual context* of a user. The individual user context is the set of information about the user and his environment which are relevant to the specific user. This means if person A is interested in the current location of person B then this information is part of the individual context of user A and is stored on A's device. The location of person C which is not of interest to person A is therefore not part of person A's context.

The concept of an individual user context also has impact on the context model itself. In order to be able to exchange only the information a user is interested in the context model provides type definitions for small, autonomous context items. These context items represent atomic information which are exchanged by the users. The context items are linked, making it possible to access other context items by following these links.

2.2 The meeting system's context model

Figure 1 shows an extract of the context model of the meeting system. The model consists of small classes each representing a context item type. A context item of type *Person* has the properties forename and surname and is linked with other context items, e. g. an activity or a room. The end names of the associations are used to access the linked context items.

When exchanging a context item, only the atomic item itself is transferred. Other items linked to this context item are not exchanged automatically by the system. This facilitates the transfer of a context item of type *Person*, representing a person p, and not the linked context items, e. g. p's current location. So this model provides a suitable way to achieve the intended distributed storage of individual user contexts.

But *meta information* about the context are not contained in the model yet. First of all, links between context items are mostly *time variant*. The current location of a person changes frequently which means that the known location of a user might not be up-to-date. The office association between *Person* and *Room* is also not permanent, it can change if a person moves to a different office. But in comparison to the current location of a person the office of a department member changes sparely. The validity of the links between context items is a meta information of the context, a

property of the association connecting the item types.

The *source of the context item*, which is a measure of information quality, is another important aspect. Context items provided by users have a good quality based upon the assumption that users do not generate faulty information on purpose. E. g. the current activity of a meeting system's user can be set by the user himself, indicating that the user is responsible for the set activity. Context items generated by sensors have a worse quality than user provided items, because sensors are generally not obeyed by humans. A sensor can provide the temperature associated with a certain location, the place of the sensor. But context items can also be derived from other items. In the meeting system the activity of a person can also be derived from the schedule of the person: if a person is attending a meeting then the activity of the person is *busy*. The reliability of the way of deriving a context item, the derivation rule, also influences the quality of the derived item.

Privacy is an aspect which is essential for context-aware systems. The user context can contain personal information which are not intended to be known to other users. In the meeting system the meetings notes taken by a user can contain private information which should just be accessible by the participants of the meeting. This means that access rights should be contained in the context model, indicating which access control mechanism should be used for linked items.

Several types of access rights for context items can be distinguished. The access to some context items needs not be restricted such that all users in the system can achieve these items. But some context items should only be used by applications running on the user's device, i.e. used by the owner of the item, and not be accessible by other users. A well suited example for such private context items are credit card data which should be handled with care. Some context items should be available to members of a group. In the meeting system access to the current activity of a person could be restricted to a group, e.g. all members of a department, such that the activity of a department member is only known to his colleagues.

Group access rights require the management of group memberships which sometimes is not adequate. A less complex way of restricting access can be achieved by user-dependant access rights. This means that the owner of a context item explicitly grants access to a context item depending on the identity of other users.

2.3 Modeling approach reviewed

As depicted in figure 1 the context can be modeled using a UML class diagram. It is also possible to denote the characteristics of context, e. g. the access rights, in the context model by using comments. Derivation rules can be specified by adding constraints to model elements and derived context items can be notated the UML way with a preceding "/", like the derived activity of a person in the meeting system. But the deficit of this approach is that comments are misused and the semantics of the comments remain unclear. The resulting models can also be invalid, e. g. if does not contain a derivation rule for every derived association. Also, the context modeler is not forced to model the context as intended using small, connected context items.

This restriction can be enforced by defining a UML profile which can be applied to UML models and restricts the ways how to use the UML. UML profiles are supported by a huge number of modeling tools such that the definition of a UML profile for context models is an interesting option to support the context modeler.

3 CMP: The Context Modeling Profile

3.1 The UML profile mechanism

A profile provides the ability to tailor a MOF (Meta object Facility, [8]) based metamodel for different platforms or domains by extending metaclasses from the metamodel. This means that the UML metamodel, which is based on MOF, can be specialized by a profile to adapt it to the domain context modeling. A profile is a lightweight extension of a metamodel, i.e. the specialized semantics must not contradict the semantics of the metamodel. But a profile provides ways to introduce a special terminology for a domain, to add semantics to metamodels and, which is vital, to restrict the way how to use the metamodel.

Basically, a profile is a package which contains stereotypes and constraints. A stereotype defines how an existing metaclass of the metamodel may be extended. A stereotype can be applied to those model elements in an UML model which are instances of the metaclass the stereotype extends. A profile can also provide required extension associations between a stereotype and its metaclasses. This results in the application of the stereotype to every instance of the stereotyped metaclass in the package or model the profile is applied to. Stereotypes can also own properties which are referred to as tag definitions. Tag definitions allow the specification of values, called tagged values, for all model elements the stereotype is applied to.

In general, stereotypes are used to define a domain specific terminology. To enrich the semantics of a UML model and restrict the usage of a metamodel the profile has to provide constraints which can be specified by

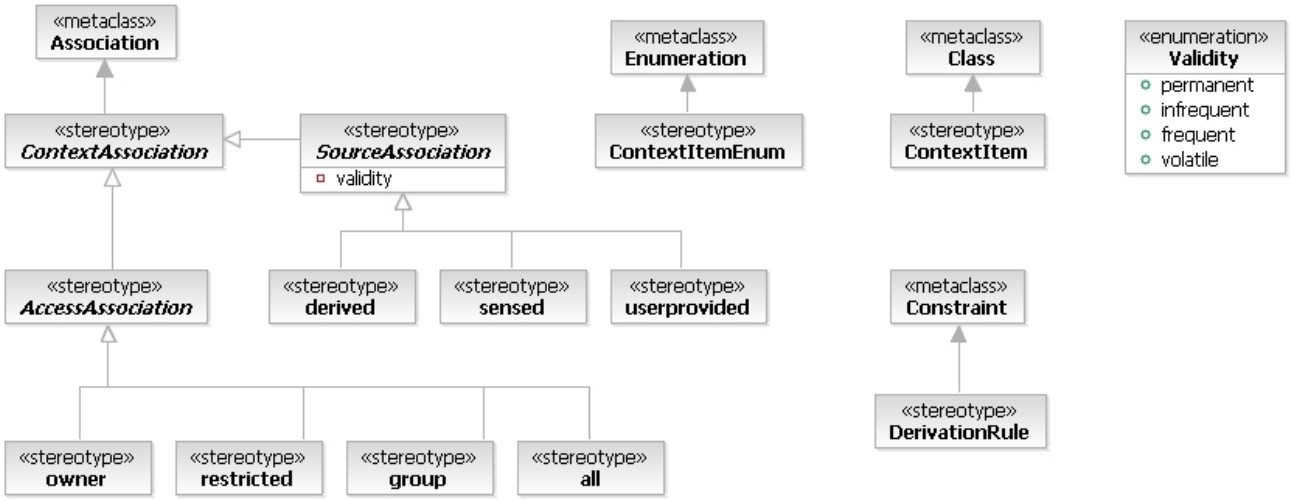

Figure 2. The context UML profile

using the OCL (Object Constraint Language, [9]). A profile constraint is a well-formedness rule, which is checked when the profile is applied to the model. A constraint represents a condition, which can be added to a model element, stating that an instance of the model element must hold this condition. Therefore, a constraint added to a stereotype must be fulfilled by all model elements the stereotype is applied to.

But certain restrictions exist in how a profile as a lightweight extension mechanism can adapt a metamodel. The constraints contained in the profile must be more restrictive than (but consistent with) the constraints of the metamodel, i.e. a profile may only specialize a metamodel. The metamodel must be extended by the profile without changes. New metaclasses must not be inserted in the class hierarchy of the metamodel and it is forbidden to modify the class definitions in the metamodel. It is therefore not allowed to have associations between stereotypes or stereotypes and metaclasses in profiles, because this is a modification of metaclasses [7]. But a similar result can be achieved by using constraints in the profile, which will be seen later on.

3.2 Stereotypes of CMP

As illustrated in the meeting system, the context model consists of the types of the context items in the user context. These classes are connected with associations which provide meta information about the context, i.e. source, validity and access rights. One can define a stereotype *ContextAssociation*, extending the metaclass *Association*, with three tag definitions, each representing one meta information. When applying the stereotype *ContextAssociation* the modeler can provide values for these tag values. In the meeting system the modeler could apply the stereotype to the association between *Person* and *Activity* and set the source property to a value like *userprovided*. But this approach has one deficit: in current UML modeling tools the tagged values are not shown in the diagram, they can just be set and retrieved by accessing property dialogs of model elements. So the meta information of the context are not part of the graphical context model, which in some way downsizes the model's expressiveness.

In order to stress the importance of privacy aspects and the source of context items as an indication of context quality, we developed CMP as depicted in figure 2. The specification of many stereotypes, each representing one source type or access right, results in the graphical annotation of the stereotype names at associations these stereotypes are applied to. The profile abstains from using required extension associations between stereotypes and metaclasses, making it possible to apply the stereotypes to designated and not to all elements of a model or package.

CMP contains the stereotype *ContextItem* extending the metaclass *Class*. This enables the context modeler to apply this stereotype to classes representing types of context items like the class *Person* in the meeting system. The stereotype *ContextItemEnum*, extending *Enumeration*, can be used to model context item types where only a predefined set of context items exists, like the type *Activity* in the meeting system.

The stereotype *ContextAssociation* extends the metaclass *Association*. This stereotype is abstract such

that it can not be applied to associations in the profiled context model. The existence of *ContextAssociation* has a technical background, because it eases the specification of OCL constraints which should restrict all stereotyped associations in the context model.

The source of context items is represented by the abstract stereotype *SourceAssociation* which specializes *ContextAssociation* and therefore does not need to extend a metaclass. The stereotypes *userprovided*, *sensed* and *derived* are all specializations of *SourceAssociation* and hence can be applied to associations. The stereotype *userprovided* indicates that the user provided the context item himself, *sensed* denotes context items taht are provided by sensors and *derived* is intended for associations between context items which are derived from other context items. The derivation rules can be expressed in the context model by using OCL constraints and the stereotype *DerivationRule*.

The validity degree of links between context items is not represented by stereotypes but by the property *validity* of the stereotype *SourceAssociation*. Hence, the modeler can specify validity of an association after applying a stereotype like *userprovided*. This way of representing the validity is founded on the design decision that the validity is an important meta information of the context and must therefore be part of the model, but needs not to be part of the graphical context model. The graphical context model should only stress the source and the access rights of context items.

Four different validity degrees for links between context items are distinguished, provided by the enumeration *Validity*. Associations between context items which are never altered like the birth date of a person can be denoted with the validity *permanent*. In the meeting system, the office of a department member may change, but does not change often, hence the validity *infrequent* should be used by the modeler. The current location of a user represents a *frequent* changing information and the current system time is a *volatile* information which is out-of-date right after acquisition.

Privacy of context is vital for acceptance of context-aware applications and often neglected in context models. Hence, the access rights of context items are realized by stereotypes, making them easily visible in the context model. The stereotype *AccessAssociation* is abstract and has only technical purposes. The applicable stereotypes to express access rights in context models are *owner*, *restricted*, *group* and *all*. The stereotype *owner* is used to model private context items like the credit card example mentioned previously. Associations with an applied stereotype *restricted* indicate user-dependent access, while the stereotype *group* denotes access for members of a group. The stereotype *all* indicates no access restriction.

3.3 Well-formedness rules of CMP

Up to this point, the profile only contains stereotypes which provide the terminology of the context modeling domain. The well-formedness of a context model can only be accomplished by adding constraints to the stereotypes [4]. These constraints are evaluated by modeling tools when the profile is applied to a context model or when the modeler triggers a model validation. Hence, the constraints can be used to force context modelers to respect the rules of modeling context. In the following, the rules for the different elements of our context profile are discussed in more detail.

The stereotype *ContextItem* should be applied to a class representing the type of a context item. The following rules have to be respected when modeling a context item type:

CI1 All attributes of a context item must have a primitive type (like *String* or *Integer*) or a type representing another context item type, i.e. a type with an applied stereotype *ContextItem* or *ContextItemEnum*.

CI2 All attributes of a context item which do not have a primitive type must be members of an association, i.e. the context item type and the type of the attribute are connected by an association.

CI3 Exactly one access right stereotype and exactly one source stereotype must be applied to all associations between the context item type and other types.

CI4 The stereotype *ContextItem* must be applied to all super types of a context item type. A context item type can therefore not inherit functionality or attributes from types that do not represent a context item type.

CI5 The context item type must possess exactly one derivation rule for every derived attribute.

The abstract stereotype *ContextAssociation* is used for technical purposes to restrict all associations between context item types in a context model. The following restrictions must be accounted for:

CA1 Associations must only be used to connect context item types, i.e. types with an applied stereotype *ContextItem* or *ContextItemEnum*.

CA2 Every association in the context model must possess exactly one access right and exactly one source stereotype.

The constraint CA2 seems redundant, because CI3 already restricts the associations between context item types. But CA2 assures that a stereotype extending *Association* may only be applied to associations between context item types. Removing CA2 from the profile makes it possible to apply the stereotype to any association, also associations between types which do not represent context item types, which is not intended. In order to validate a context model, i.e. check that it fulfills the restrictions stated above, these constraints have to be expressed in OCL and must be added to the stereotypes in the profile.

Links from context items to *derived* context items must be modeled by applying the stereotype *derived* to the associations. In addition to this, the attribute must also be marked as derived. The following rule must be met:

CA3 At least one member of the association has to be derived.

The following rules must be obeyed when modeling derivation rules:

DR1 A derivation rule must constrain only one element.

DR2 The language of the constraint must be OCL.

DR3 The constrained element of a derivation rule must be a class, possessing a derived attribute. The name of the derivation rule must match the name of the derived attribute.

DR4 The constrained element of a derivation rules must represent a context item type.

The rule DR3 can be easily expressed in OCL as follows:

```
self.constrainedElement->
 exists(elem : uml::Element |
  elem.oclIsTypeOf(uml::Class) and
  elem.oclAsType(uml::Class).ownedAttribute->
   exists(attribute : uml::Property |
    attribute.isDerived and
    attribute.name = self.name
   )
 )
```

The OCL constraint above is checked for every constraint in the context model, the stereotype *DerivationRule* is applied to. The constraint checks if the set of constrained elements, that every constraint owns, contains at least one element, that is an instance of the UML metaclass *Class*. The set of properties owned by this class, which can be accessed by the attribute *ownedAttribute*, must contain at least one property with the attribute isDerived and has the same name as the derivation rule. Because *Constraint* is the metaclass of *DerivationRule*, it is possible to make use of the properties of the metaclass *Constraint* as shown above.

The OCL expression for the rule CI5 can be defined as follows:

```
let stereot : String =
  'ContextProfile__DerivationRule' in
self.ownedAttribute->
 forAll(attribute : uml::Property |
  attribute.isDerived implies
   self.ownedRule->
    exists(constraint : uml::Constraint |
     constraint.name = attribute.name and
     constraint.eAnnotations->notEmpty() and
     constraint.eAnnotations->
      exists(annot |
       annot.source = 'appliedStereotypes'
       and annot.contents->
        exists(c | c.eClass().name = stereot)
      )
    )
 )
```

This OCL constraint is not completely based on the UML metamodel. The problem is that there is currently no unique way in UML modeling tools to query the applied stereotypes of model elements, which is necessary to restrict the combined usage of stereotypes. The context profile was developed using the IBM Rational Software Architect, which is based upon EMF (Eclipse Modeling Framework). Hence, functionality of EMF is used in the context model to query the applied stereotypes.

The other well-formedness rules stated above can be expressed in OCL in a similar way. The OCL constraints defined in CMP make associations between stereotypes obsolete. The same effect is achieved by querying the applied stereotypes of model elements. None of the metaclasses of the UML had to be modified, such that CMP is a lightweight extension of the UML metamodel. The approach of checking the applied stereotypes of model elements to restrict the usage of the UML is the proposed way to circumvent the modification of classes of the UML metamodel.

4 Application of CMP

The application of CMP is illustrated using the meeting system case study. Figure 3 shows the same extract of the context model of the meeting system as figure 1, but the profile has been applied to the model.

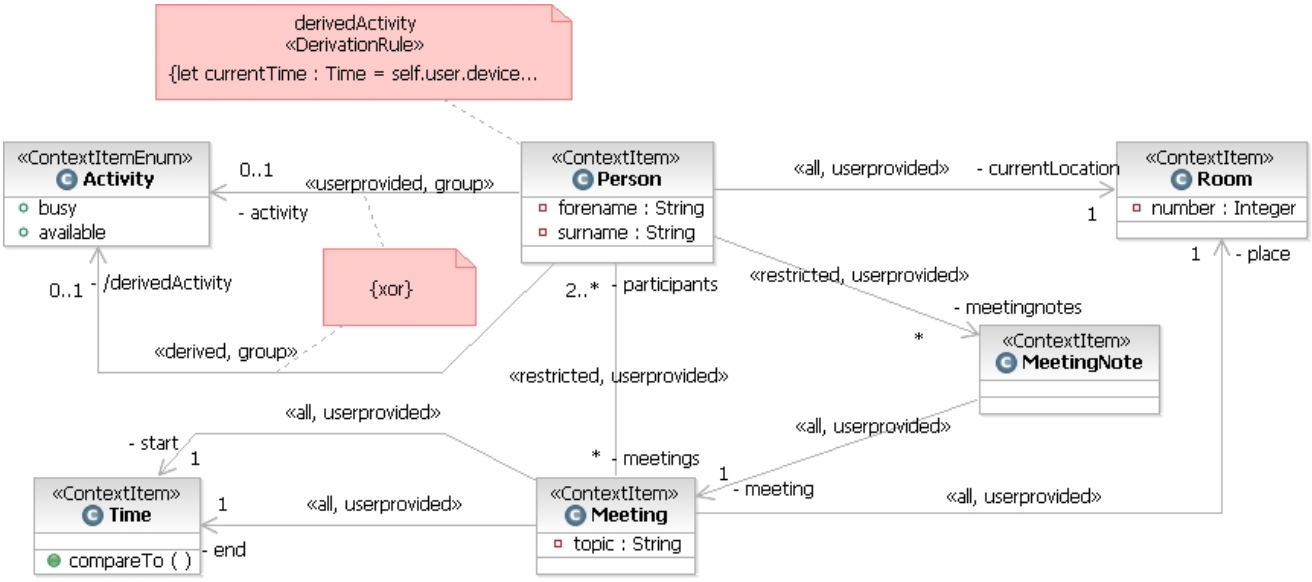

Figure 3. The meeting system after applying CMP

The stereotype *ContextItem* is applied to every type in the context model except for *Activity*. The different activities of a user are a predefined set of context items, hence it is modeled as an enumeration and the stereotype *ContextItemEnum* is applied to the type.

It is obvious that every association has exactly one applied stereotype indicating the source and one stereotype indicating the access rights. E.g. the current location of a person, the room in the department the user is currently located at, is provided by the user and can be accessed by all users in the system. The current location of a user changes frequently, hence the validity property of the association, which is provided by the applied stereotype *userprovided*, is set to *frequent*. But as mentioned earlier, the values of properties of stereotypes, the tagged values, are not shown in the graphical model such that the validity of the current location of person can only be retrieved by accessing the properties of the association.

Every meeting in the set of meetings of a person is provided by a user. The schedule of department members should be automatically matched by the system, but the appointment must be acknowledged by the user. Hence, the user is responsible for the correctness of the context items, making it feasible to apply the stereotype *userprovided* to the association. The stereotype *restricted* is also applied, indicating that the owner of the context items, i.e. the user who owns the device the context items are stored on must manually grant access. Because a meeting should just be accessible for participants of the meeting, a manual access decision, denoted by *restricted*, based on the user identity is more appropriate than the management of a user group for every single meeting.

For the same reasons the stereotypes *userprovided* and *restricted* are also applied to the association between *Person* and *MeetingNotes*. But the association between *MeetingNotes* and *Meeting* is noticeable, making it possible to access the associated meeting of the notes by following the corresponding link between the items. As shown in figure 3, this association is not restricted, which is indicated by the applied stereotype *all*. This means that if a user has access to meeting notes then he can also access the associated meeting without any check of access rights. This implies that if and only if a user in the system has access to a context item representing meeting notes of a meeting, he can also access the associated meeting.

A person in the meeting system has two links to an activity context item. One can be accessed by *activity*, the other one by *derivedActivity*. The *derivedActivity* is modeled by setting the attribute *isDerived* of the association end, indicated by the preceding "/". In figure 3 an *xor*-constraint is annotated to the associations, expressing that either the derived activity or the activity provided by the user is set, but not both. The *xor*-constraint is a predefined constraint of the UML.

The derived activity of a person is an attribute of the class *Person*. According to the rules of context modeling discussed in the previous section, a derivation rule with the name *derivedActivity* is provided, which constrains context items of type *Person*. The derivation

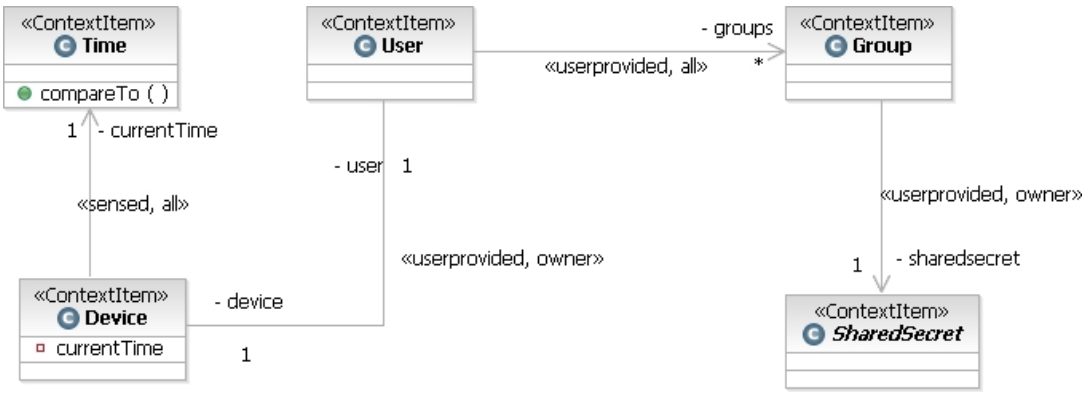

Figure 4. Modeling of user groups

rule contains the following OCL expression:

```
let currentTime : Time =
  self.user.device.currentTime in
self.meetings->
 exists(meeting : Meeting |
  meeting.start.compareTo(currentTime) <= 0
  and meeting.end.compareTo(currentTime) > 0
 )
implies self.derivedActivity = Activity::busy
```

The above OCL expression states, that if the user is attending a meeting, which is derived from the current system time and the time attributes of the user's meetings, then the activity of the user can be set to *busy*. Modeling environments support the definition of OCL constraints and automatically verify, that all constraints of a model are syntactically valid. E. g. Hence, a modeler is notified if an unspecified or deleted property of a class is used in a constraint definition. Only the syntax of constraints can be checked, because a constraint does not restrict the model elements like a class, but the instances of the elements, e. g. an object.

User groups, which must be used, when group access rights are set in a context model, can also be part of the context model as shown in figure 4. The basic idea is that every user in the system is member of certain groups. All group members possess the shared secret of the group, making it possible to verify, whether another user is also a member of the group. The shared secret, represented by the abstract context item type *SharedSecret*, can only be accessed by the owner of the device, similar to the previously mentioned credit card example. Hence, the shared secret can not be automatically distributed among the users in the system, but must be exchanged using a different channel, hence it is denoted as *userprovided*.

5 Related Work

Of course, other approaches for modeling context exist, most of them based on a designated modeling language. Henricksen et al. developed CML (Context Modelling Language, [5]), which is an extension of ORM (Object-Role Modeling). Validity and quality of context entities are regarded and derivation rules can be expressed by using predicate logic. But the approach has several drawbacks: CML is appropriate for describing context models for distributed systems with a designated central storage node. But according to our understanding of privacy the user must have control of the distribution of private context items, which can hardly be achieved with a central storage node. The authors also propose a scheme-based ownership-notation [6], by which privacy preferences should be realized. But the modeling of these preferences is left out. The lack of integration of visual context model, logical rules and scheme-based ownership-notation can also arise consistency problems.

Several ontology based context modeling languages exist, like CoOL by Strang et al. [12], the COBRA ontology by Chen and Finin [3] or CONON by Wang et al. [14]. These modelling languages are all based on OWL (Web Ontology Language, [13]) and provide a basic context model which can be extended. Some of these approaches lack of a metamodel for context models, limiting the modeler to context models which extend the basic model. The major benefit of an ontology-based approach is that reasoning about context is possible. Reasoning requires computational power, which can not be provided by PDAs. Hence, these context modeling languages have a different application domain: they are intended for service provision based on Web Services, intelligent houses or multi-agent systems. Reasoning also imposes new privacy issues, be-

cause information about the user can be gained without the user's knowledge. Most of these languages model do not address privacy and impose system requirements which contradict our assumptions about distributed mobile systems.

Sheng and Benatallah developed Context UML [11], which is intended to be used for the development of context-aware Web Services. The authors provide a metamodel for modeling context, but mainly focus on the modeling of Web Services. Context UML allows the modeling of derivation rules, but does not include means to model user privacy. The authors provide a heavyweight extension of the UML metamodel by modifying the UML metaclasses. This approach has the drawback that the extension can not be used by UML modeling tools.

6 Conclusions and Future Work

This paper showed that CMP restricts the usage of the UML such that the UML is successfully adapted to the domain of context modeling. We provided a lightweight extension of the UML without modifying the UML metamodel, which is rarely achieved by other authors. This approach enables CMP to be integrated in many UML modeling tools, supporting context modelers with the modeling tool they are used to.

UML models can consist of a set of diagrams, each focusing a special aspect of a model, making the model easier to understand. The clear distinction between visual representation of the model and the model itself, which is provided by every UML modeling tool, allows consistent modifications of model elements, improving the development of context models. UML models can also be exchanged by using XMI (XML Metadata Interchange) documents, making it possible to import the model into a different modeling environment.

Another benefit of the UML is the support of OCL: logical rules can be integrated into the model. The constraints are not specified in addition to the context model or coexist with the model, but are part of the model. Current tools support the parsing of OCL expressions such that invalid OCL expression can be detected on modeling level, i.e. the usage of incorrect types or properties can easily be detected. The transformation of OCL expressions into programming languages like C# or Java is also supported by modeling tools or can be achieved by external tools.

The constraints of CMP depend on EMF, because a general way to access the applied stereotypes of a model element is not supported by modeling tools yet. This seems to be a flaw in the UML metamodel. When a general way is provided, the constraints will be adapted. But this will not result in a change of the semantics and usage of CMP such that the modification of the OCL constraints will not be noticed.

Currently, the context UML profile is applied to other case studies to verify, whether further modeling rules can be stated or not. Also, the functionality of the meeting example is enhanced to verify that the resulting models are still easy to understand when dealing with more complex scenarios. But our approach of application specific context models seems to be a more promising way than the usage and refinement of huge, predefined models as proposed by other authors. Another result of these case studies will be a small, basic context model, as indicated in figure 4.

Further stereotypes will be added to the profile which can be used to model dynamic behaviour, i.e. contextual situations and adaption strategies, which are necessary to model user triggered and automatic adaption of context-aware applications. As soon as the profile is released, tools for the transformation of a profiled context model into programming languages and an ontology will be implemented, supporting the development of context-aware applications with UML. The overall goal of this work is the provision of a framework for context-aware applications in a mobile, distributed systems with respect of the user privacy.

References

[1] G. D. Abowd and A. K. Dey. Towards a better understanding of context and context-awareness. In *Workshop on The What, Who, Where, When, and How of Context-Awareness (CHI2000)*, volume 1707 of *Lecture Notes In Computer Science*, pages 304–307. Springer, September 2000.

[2] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Department of Computer Science, Dartmouth College, November 2000.

[3] H. Chen, T. Finin, and A. Joshi. An ontology for context-aware pervasive computing environments. Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, 2003.

[4] M. Gogolla. Using OCL for defining precise, domain-specific UML stereotypes. In A. Aurum and R. Jeffery, editors, *Proc. 6th Australian Workshop on Requirements Engineering (AWRE'2001)*, pages 51–60. Centre for Advanced Software Engineering Research (CAESER), University of New South Wales, Sydney, Australia, 2001.

[5] K. Henricksen and J. Indulska. A software engineering framework for context-aware pervasive computing. In *Proc. 2nd IEEE Conf. on Pervasive Computing and Communications*, pages 77–86. IEEE Computer Society, Orlando, USA, 2004.

[6] K. Henricksen, R. Wishart, T. McFadden, and J. Indulska. Extending context models for privacy in pervasive computing environments. In *Context Modelling and Reasoning Workshop at PerCom 05*, pages 20–24. IEEE Computer Society, Kauai Island, Hawaii, 2005.

[7] Object Management Group. UML 2.0 superstructure specification, formal/05-07-04.

[8] Object Management Group. Meta object facility (MOF) 2.0 core specification, ptc/2005-06-06.

[9] Object Management Group. OCL 2.0 specification, ptc/2005-06-06.

[10] B. N. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, USA, 1994.

[11] Q. Z. Sheng and B. Benatallah. ContextUML: A UML-based modeling language for model-driven development of context-aware web services. In *Proceedings of the International Conference on Mobile Business (ICMB'05)*, 2005.

[12] T. Strang, C. Linnhoff-Popien, and K. Frank. Cool: A context ontology language to enable contextual interoperability. In J.-B. Stefani, I. M. Demeure, and D. Hagimont, editors, *DAIS*, volume 2893 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 2003.

[13] W3C. OWL Web Ontology Language.

[14] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology based context modeling and reasoning using owl. In *Context Modeling and Reasoning Workshop at PerCom 04*, pages 18–22. IEEE Computer Society, 2004.

[15] M. Weiser, R. Gold, and J. S. Brown. The origins of ubiquitous computing research at parc in the late 1980s. *IBM Systems Journal*, 38(4), pages 693–696, 1999.

Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)
0-7695-2755-8/07 $20.00 © 2007 IEEE