

Higher-Order Entity Relationship Modelling with UML

Alexei Tretiakov and Sven Hartmann
Department of Information Systems, Massey University
Palmerston North, New Zealand
[A.Tretiakov | S.Hartmann]@massey.ac.nz

Abstract

To explore the use of UML for creating value-representable data models, we use the Object Management Group (OMG) Meta Object Facility (MOF) to formally describe the Higher-order Entity Relationship Model (HERM). In addition, we formally define a UML profile for HERM. By demonstrating that MOF and UML can represent the most critical capabilities of HERM, we open way for better integration of HERM models into projects relying on UML for application modelling and for the use of UML tools for HERM modelling.

1. Introduction

A persistently recurring theme in literature devoted to software quality is the ability of methods relying on mathematical proofs (often referred to as formal methods) to guarantee certain quality aspects. Critiques of the use of methods ultimately based on mathematics argue that they increase the cost of development and are incomprehensible to clients (see, e.g. [2]). One possible approach to make formal approaches more accessible is to disguise the rigor and the associated complexity by providing suitable metaphors and graphical notation. Indeed, one of the most spectacular successes of mathematically sound approaches is the dominant role of the Relational Data Model (RDM) in database development. We argue that it is to a large degree due to the metaphor of a table used by the Relational Data Model.

In this article we attempt to create a familiar “user interface” for another data model with strong mathematical foundations: the Higher-order Entity Relationship Model (HERM), introduced by Thalheim [10]. To apply a UML class diagram as a representation of HERM, we redefine HERM within the UML standard, but at a level of formality higher than it is a normal practice in UML modelling and in the UML specification [7, 8] itself.

Despite the importance of the Entity Relationship (ER) Model, with applications ranging from business analysis

and relational database design to ontologies and linguistics, there is no widely accepted standard for ER model notation and semantics. Over the years, a number of variations and extensions of the basic ER Model were introduced. Features of most of these extensions are captured by the Higher-order Entity Relationship Model (HERM), which rests on solid mathematical foundations, and was demonstrated to result in more semantically rich, more compact, but easier to understand data schemas than the basic ER Model [10].

The field of object-oriented (OO) modelling of application structure relies on the widely accepted Unified Modelling Language (UML) standard maintained by the Object Management Group (OMG) [7, 8]. This standard is supported by a range of high profile tool vendors. It includes a built-in extension mechanism, provides a standard XML schema for metadata exchange (XMI), and offers a standard approach to generating CORBA interfaces for metadata management (with likely future extensions to support Java RMI and XML services).

The major difference between UML class diagrams and ER modelling is that UML class diagram modelling relies on implicit object identity. In UML, objects do not need to be value-representable: distinct objects with exactly the same attribute values and the same links to other objects are allowed, because objects rely on implicitly present object IDs for their identity. On the other hand, the ER Model and its extensions (including HERM) rely on value-representability for object identity: each object can be retrieved by using a query involving attribute values only. These can be attributes of the object itself, and attributes of directly or indirectly related objects. In the ER Model and its extensions, relations between objects are established via attribute values (“set semantics”), rather than via implicit object IDs (“reference semantics”). As a result, a number of important constraints (such as primary key and inclusion constraints) are graphically represented in ER diagrams, but not in UML class diagrams.

Due to the widespread availability of UML tools, it would be beneficial to use UML or its extensions for data modelling, thus achieving better integration of artifacts into

projects and better utilisation the existing expertise, and benefiting from the standard infrastructure provided by the UML specification, such as XMI DTDs and CORBA IDL generation. While attempts were made to use UML class diagrams for data modelling, they were effectively limited to extending UML (formally, by using the UML extension mechanism, or informally) by adorning UML classes to represent ER Model entities, and to adorning UML associations to represent ER Model relationships [3, 4, 5]. This approach cannot be used with HERM because the UML specification does not allow associations as participants for association ends [7, 8], so that relationships between relationships (an important feature of HERM) cannot be represented this way.

In this article, we explore two approaches to incorporate HERM into the UML infrastructure. Firstly, we use the Meta Object Facility (MOF) [6] to define HERM as an M2 metamodel. This allows adding HERM to the UML standard as a separate UML metamodel along with UML class diagrams and other standard UML metamodels. In addition, the MOF metamodel allows us to unambiguously restate HERM syntax (corresponding to allowed M1 layer instances) and semantics (corresponding to allowed M0 layer instances). Secondly, as an alternative, more practically oriented approach, we formally define HERM as a package extending the UML class diagram notation, and demonstrate that the resulting syntax and semantics incorporate all features introduced by using MOF.

The advantage of the first approach is its clarity: the resulting metamodel has the features of HERM, and does not inherit any additional features. The advantage of the second approach is that it may allow creating HERM database schemas by using existing UML tools. Thus one can potentially benefit from having access to class diagram features such as operations and methods while preserving the semantic richness of HERM database schemas, and allowing value-representable and conventional classes to coexist in the same diagram.

2. MOF metamodel for HERM

2.1. Informal overview

A MOF metamodel for HERM is given in Figure 1. Informally, objects are allocated to *vrClasses*, so that each object belongs to exactly one *vrClass*. Each object in a given *vrClass* has the same set of attributes. These attributes are given as linked *Attribute* instances, with values in domains determined by the linked *DataType* instances. In addition, each object in a *vrClass* is linked to a number of *Component* instances. A *Component* is either another *vrClass* instance or a *Cluster* instance. A *Cluster* is set of objects that

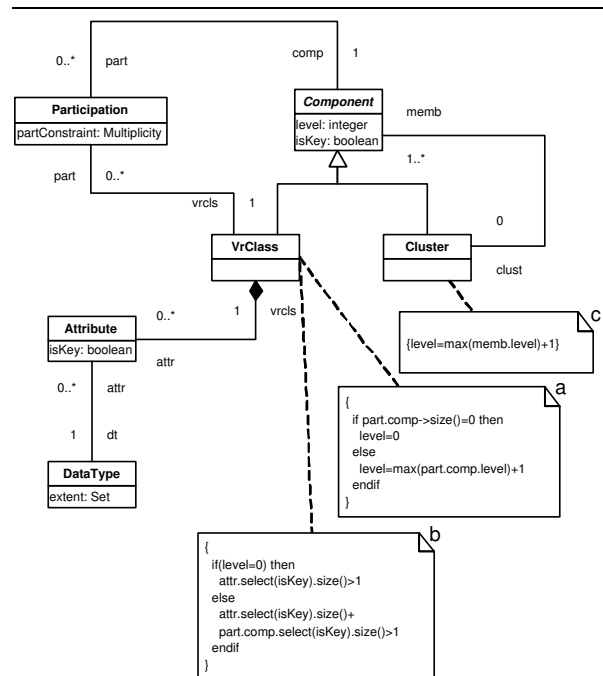


Figure 1. MOF metamodel for HERM.

is formed as the union of certain *vrClass* instances and/or other *Cluster* instances.

Value-representability is achieved by hierarchically assigning level numbers to all *vrClass* and *Component* instances. A *vrClass* instance of level 0 has no components, but is required to have at least one attribute which should be marked as a key attribute. The identity of objects in a *vrClass* instance of level 0 is determined by the values of its key attributes.

For all other *vrClass* instances, the level is determined by the maximum level of their components plus one. The identity of objects in *vrClass* instances of higher level (that is, greater than zero) is determined by the values of their key attributes, together with the identities of the linked objects in the corresponding key *Component* instances.

As each object in a *Cluster* instance also belongs to some *vrClass* instance at a lower level, its identity is already determined at the level of the *vrClass* instance. Ultimately, the identity of each object is determined by values of attributes at the same or at lower levels, which makes each object value-representable.

Finally, one should note that while the hierarchical structure imposed by restrictions of the HERM metamodel is sufficient to guarantee value-representability, value-representability is also possible in schemas with loops [9].

2.2. Layer M1 (Syntax)

Here we discuss constraints determining which model instances (in MOF terminology, M1 instances) are allowed. This relates to which models are well-formed, and thus determines the model syntax. Multiplicity constraints are shown in Figure 1 and do not require further clarification. In particular we note that while a *vrClass* instance may have zero or more components, a *Cluster* instance should have at least one member. Constraints that cannot be expressed graphically are listed below in natural language and in logic notation. In Figure 1, they are also given in Object Constraints Language (OCL). In the following, we use $vrClass(\xi)$ as a predicate that evaluates to true if ξ denotes a *vrClass* instance. Furthermore, $\xi.part(\pi).comp(\eta)$ evaluates to true if η is an M1 layer instance that can be navigated to from ξ over a Participation instance π .

A *vrClass* instance ξ without any components is of level 0. A *vrClass* instance ξ with components is one level higher than its highest level component.

$$vrClass(\xi) \wedge \neg \exists \pi, \eta. (\xi.part(\pi).comp(\eta)) \rightarrow \xi.level = 0$$

$$vrClass(\xi) \wedge \exists \pi, \eta. (\xi.part(\pi).comp(\eta)) \rightarrow \xi.level = \max_{\{\eta | \exists \pi. (\xi.part(\pi).comp(\eta))\}} \eta.level + 1$$

A *vrClass* instance ξ of level 0 should have at least one attribute, which should be a key attribute. At any level, a *vrClass* instance should have either a primary key attribute, or a primary key component.

$$vrClass(\xi) \wedge \xi.level = 0 \rightarrow \exists \alpha. (\xi.attr(\alpha) \wedge \alpha.isKey)$$

$$vrClass(\xi) \rightarrow \exists \alpha. (\xi.attr(\alpha) \wedge \alpha.isKey) \vee \exists \pi, \eta. (\xi.part(\pi).comp(\eta) \wedge \eta.isKey)$$

Similarly, we use $Cluster(\xi)$ as a predicate that evaluates to true if ξ denotes a *Cluster* instance. A *Cluster* instance is one level higher than its highest level member.

$$Cluster(\xi) \rightarrow \xi.level = \max_{\{\eta | \xi.mem(\eta)\}} \eta.level + 1$$

2.3. Layer M0 (Semantics)

Here we discuss how instances of well-formed M1 models are formed, and thus introduce their semantics.

In this section, we continue to use Greek letters to denote *vrClass*, *Cluster*, or *Component* instances. Along with them we will use lower case Latin letters to denote instances of *vrClass*, *Cluster*, or *Component* instances (M0 layer objects), which we will often call just *objects*.

As before, for a *vrClass*, *Cluster*, or *Component* instance ξ , we use $\xi(x)$ as a predicate which evaluates to true if object x is an instance of ξ (that is, object x belongs to ξ).

Let us formally define how M0 layer model instances are formed. Consider a countable domain \mathcal{D} of objects with implicit identity. An M0 layer model instance is formed by assigning a finite number objects to each *vrClass* instance in the M1 layer model, so that an object is assigned to no more than one *vrClass* instance.

$$vrClass(\xi) \wedge vrClass(\eta) \wedge \xi(x) \wedge \eta(y) \wedge \xi \neq \eta \rightarrow x \neq y$$

The extent of a *Cluster* instance ξ is the union of the sets of objects assigned to the members of ξ :

$$\{x \mid \xi(x)\} = \{y \mid \exists \eta. (Component(\eta) \wedge \xi.mem(\eta) \wedge \eta(y))\} \quad (1)$$

We observe that according to the MOF generalisation semantics, each *vrClass* or *Cluster* instance is automatically a *Component* instance, too, so that equation (1) allows to recursively determine the extent of all *Cluster* instances.

In addition, for each M0 layer model instance, the following two mappings are defined. Given a *vrClass* instance ξ , linked to an *Attribute* instance α , and an object x belonging to ξ , the mapping φ provides the value of the attribute α for that object:

$$\varphi(\xi, \alpha, x) \in \alpha.dt.extent$$

In other words, for each object, values are provided for all of its attributes.

Given a *vrClass* instance ξ , linked to a *Component* instance η via a Participation instance π , and an object x belonging to ξ , the mapping ψ provides an object from η :

$$\psi(\xi, \pi, \eta, x) \in \{y \mid Component(\eta) \wedge \xi.part(\pi).comp(\eta) \wedge \eta(y)\}$$

In other words, each object is linked objects of all components of its *vrClass*. Hence, an object of a *vrClass* instance of level greater than zero can only exist if there are objects belonging to its components, to which it is linked.

One should observe that for the same ξ and η , an object x can be mapped to different objects of η depending on the Participation instance π .

Now we are ready to introduce constraints guaranteeing value-representability. For each vrClass instance of level 0, object identity is uniquely represented by key attribute values:

$$\begin{aligned} vrClass(\xi) \wedge \xi.level = 0 \wedge \xi(x) \wedge \xi(y) \wedge x \neq y \rightarrow \\ \exists \alpha. (Attribute(\alpha) \wedge \xi.attr(\alpha) \wedge \\ \varphi(\xi, \alpha, x) \neq \varphi(\xi, \alpha, y)) \end{aligned}$$

More general, for each vrClass instance, object identity is uniquely represented by a combination of key attribute values and linked key component object identities:

$$\begin{aligned} vrClass(\xi) \wedge \xi(x) \wedge \xi(y) \wedge x \neq y \rightarrow \\ \exists \alpha. (Attribute(\alpha) \wedge \xi.attr(\alpha) \wedge \\ \varphi(\xi, \alpha, x) \neq \varphi(\xi, \alpha, y)) \vee \\ \exists \pi, \eta. (Component(\eta) \wedge \xi.part(\pi).comp(\eta) \wedge \\ \psi(\xi, \pi, \eta, x) \neq \psi(\xi, \pi, \eta, y)) \end{aligned}$$

Hence, for each vrClass instance of level 1, object identity is represented by attribute values for level 0 and level 1 objects (with attributes of level 0 objects representing object identity of level 0 objects). Therefore, the identity of objects belonging to vrClass instances of level 1 is value-representable. Similarly, for objects belonging to Component instances of level n , as it is easy to show by recursion, object identity is represented by values of attributes at level n and below. Consequently, the identity of all M0 objects is value-representable.

Finally, we define the semantics of participation constraints provided via partConstraint attributes of Participation instances. For each object x belonging to a Component instance ξ , the number of objects in a vrClass instance η linked at M1 layer to ξ that are mapped to x at layer M0 does not exceed the value of partConstraint on the corresponding Participation instance:

$$\begin{aligned} Component(\xi) \wedge \eta.part(\pi).comp(\xi) \wedge \xi(x) \rightarrow \\ \# \{y \mid \psi(\eta, \pi, \xi, y) = x\} \leq \eta.part(\pi).partConstraint \end{aligned}$$

2.4. Mapping instances of the HERM metamodel to relational models

To clarify the semantics still further we describe how HERM models can be mapped to equivalent relational models.

While each relational model can be viewed as a HERM model with all vrClass instances at level 0, and no Cluster instances, we take a simpler (yet, equivalent) approach by defining a relational database schema as a finite set of finite sets of attributes, or a finite set of relation schemas,

with each relation schema given as a finite set of attributes. Let us define a mapping R from the set of all vrClass instances to the set of all relations.

Mapping R maps vrClass instances to relation schemas representing them in the relational model as follows. Each vrClass instance of level 0 is mapped to a relation comprised of its attributes. The primary key of the resulting relation is given by the key attributes of the vrClass instance.

$$\begin{aligned} vrClass(\xi) \wedge \xi.level = 0 \rightarrow \\ R(\xi) = \{\alpha \mid Attribute(\alpha) \wedge \xi.attr(\alpha)\} \end{aligned}$$

$$\begin{aligned} vrClass(\xi) \wedge \xi.level = 0 \rightarrow \\ K(R(\xi)) = \{\alpha \mid Attribute(\alpha) \wedge \xi.attr(\alpha) \wedge \alpha.isKey\} \end{aligned}$$

For a vrClass instance linked to other vrClass instances only, the mapping is defined as follows:

$$\begin{aligned} R(\xi) = \{\alpha \mid Attribute(\alpha) \wedge \xi.attr(\alpha)\} \uplus \\ \biguplus_{\{(\pi, \eta) \mid Component(\eta) \wedge \xi.part(\pi).comp(\eta)\}} K(R(\eta)) \quad (2) \end{aligned}$$

Along with the attributes linked to a vrClass instance ξ , one also imports the attributes belonging to primary keys for the relation schemas to which the Component instances linked to ξ have been mapped. Note that the unions on the right hand side of equation (2) are disjoint, as the same attribute may enter the resulting relation schema more than once, over different Participation instances (in different “roles”). In practice, disjoint unions can be implemented by appropriately renaming the attributes. A combination of the key attributes of the vrClass instance ξ and the imported key attributes forms the primary key, thus, serving as a substitute of an implicit object ID.

A vrClass instance linked to at least one Cluster instance can not be represented by a single relation schema, but rather a set of relation schemas is required. For such a vrClass instance ξ , the set of relation schemas representing it in the relational model is built recursively as outlined below. To begin with, we construct a relation schema $R(\xi)$ that is obtained for ξ as described above by only considering the vrClass instances to which ξ is linked to. Now, let us arrange the Cluster instances to which ξ is linked to in some order, say $\eta_1, \dots, \eta_{n(\xi)}$. Start with a set of relation schemas T_0 containing a single relation schema, namely $R(\xi)$. Next, on considering the first Cluster instance η_1 , a new set of relation schemas T_1 is obtained as follows:

$$T_1 = \bigcup_{X \in C(\eta_1), Y \in T_0} \{Y \uplus K(X)\}$$

Herein, $C(\eta)$ is a set of all vrClass instances that are directly or indirectly (via a sequence of linked cluster instances) members of η :

$$C(\eta) = \{\nu \mid vrClass(\nu) \wedge (\eta.mem(\nu) \vee \exists \nu'. (Cluster(\nu') \wedge \eta.mem(\nu').mem(\nu)) \vee \dots)\}$$

The sets T_2 to $T_{n(\xi)}$ are obtained recursively, similarly to T_1 :

$$T_2 = \bigcup_{X \in C(\eta_2), Y \in T_1} \{Y \uplus K(X)\}$$

$$\dots$$

$$T_{n(\xi)} = \bigcup_{X \in C(\eta_{n(\xi)}), Y \in T_{n(\xi)-1}} \{Y \uplus K(X)\}$$

Eventually, the set of relation schemas $T_{n(\xi)}$ represents the vrClass instance ξ in the relational model.

It should be noted, that Cluster instances contribute to the relational model representation only as Component members, not directly. The reason for that is that Cluster instances that are not (directly or indirectly) Component members do not impose any restrictions on the allowed M0 layer instances. In a relational database implementation they can be represented as views, however, as in the basic relational model relations can not be specified as being derived from other relations, they have to be left out.

3. UML profile for HERM

The MOF metamodel introduced in the previous section allows one to get a deep insight into HERM syntax and semantics. However, since HERM models are in many respects similar to UML class diagrams, it makes sense to introduce HERM as a UML profile based on UML class diagrams by using the UML extension mechanism, in particular, stereotypes, tagged values and constraints associated with stereotypes. Comparing to the previous section, in this section we adopt a less formal treatment. The stereotypes used in the UML profile for HERM are summarised in the Table 1.

It is easy to see that in conjunction with the standard UML syntax, the stereotypes defined in Table 1, together with their constraints, define a syntax that is consistent with the MOF metamodel defined in Section 2: vrClass instances are mapped to $\ll vr \gg$ classes, while Cluster instances are mapped to $\ll cluster \gg$ classes. The model semantics defined in Section 2 does not contradict the standard UML semantics implied by the profile defined in Table 1, but defines the semantics more precisely than the UML standard.

Stereotype	vr
Base class	Class
Tags	level: Integer
Description	A value-representable class. Connected via $\ll part \gg$ associations to $\ll vr \gg$ or $\ll cluster \gg$ classes.
Constraints	<ul style="list-style-type: none"> The value of level should be the maximum level of the $\ll vr \gg$ and $\ll cluster \gg$ classes associated to it as aggregates via $\ll part \gg$ associations, plus 1. The value of level should be 0 if it does not enter any $\ll part \gg$ association opposite to the sharable aggregation end. A level 0 $\ll vr \gg$ class should have at least one $\ll data \gg$ attribute with key tag value set to true. At any level, a $\ll vr \gg$ class should either enter at least one $\ll part \gg$ association opposite to the sharable aggregation end with the aggregation's tagged value key set to true, or it should have at least one $\ll data \gg$ attribute with key tag value set to true.
Stereotype	cluster
Base class	Class
Tags	level: Integer
Description	A cluster. Represents the union of its members, that is, the $\ll vr \gg$ and/or $\ll cluster \gg$ classes it generalises.
Constraints	<ul style="list-style-type: none"> Should be an abstract class with no $\ll data \gg$ attributes. Should be on the parent side of at least one $\ll memb \gg$ generalisation. The value of level should be the maximum level of the $\ll vr \gg$ and $\ll cluster \gg$ classes it generalises, plus 1.
Stereotype	part
Base class	Association
Tags	key: Boolean
Description	Connects $\ll vr \gg$ classes to their components.
Constraints	<ul style="list-style-type: none"> Should be connected to a $\ll vr \gg$ class at one end (for which the multiplicity should be set to represent the participation constraint), and to a $\ll vr \gg$ or $\ll cluster \gg$ class at another end, for which the aggregation property should be set to sharable.
Stereotype	memb
Base class	Generalisation
Description	Connects $\ll cluster \gg$ classes to their members.
Constraints	<ul style="list-style-type: none"> Should have a $\ll cluster \gg$ class as a parent, and a $\ll vr \gg$ or $\ll cluster \gg$ class as a child.
Stereotype	data
Base class	Attribute
Tags	key: Boolean

Table 1. UML profile for HERM.

As to visual presentation of model elements adorned by stereotypes defined in Table 1, while UML specification offers customisation hooks, we prefer to leave a high usability visual presentation as a topic for further research. We note however, that there is strong anecdotic evidence that visual presentation adopted in [10] is not entirely satisfactory, as it relies on applying the metaphor of a “relationship” to constructs that are not recognised as relationships in everyday language (e.g., relationships with only one participant in a single role). Indeed, the most widely known design tool using HERM, Kleen modeler [1] does not distinguish “entity” and “relationship”, but views both as “assets”. Some examples of UML rendition of HERM diagrams (using standard UML graphical representation) are given in Figures 2 to 5. Figures 2, 3 and 4 demonstrate some typical use of higher level vrClass instances (in a sense, they introduce some basic HERM design patterns). Figure 5 is a more full-featured example.

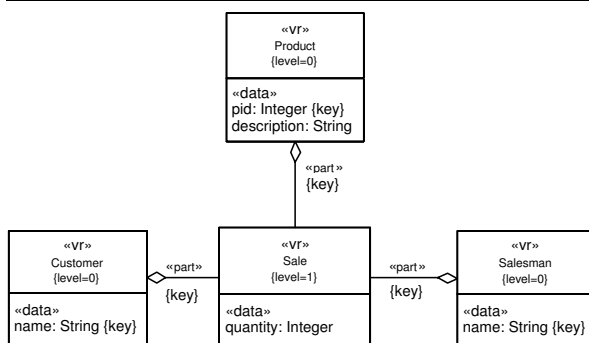


Figure 2. A higher level vrClass instance acting as a relationship.

In Figure 2, the vrClass instance Sale effectively acts like a relationship in traditional ER notation. Intuitively, Customer relates to Salesman (and Product) as all of them share instances of Sale (or participate in instances of Sale).

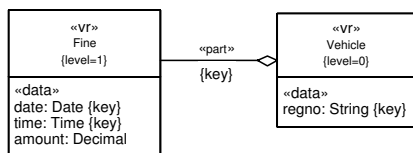


Figure 3. Higher level vrClass instance acting as an aggregate part.

In Figure 3, the higher level vrClass instance Fine, rather than acting as a relationship, forms an aggregate with Vehicle by “attaching” itself to it. Any number of Fine objects can be attached to a Vehicle object. In the context of UML, this interpretation is of course straightforward.

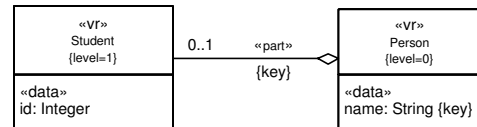


Figure 4. A higher level vrClass instance acting as a specialisation of a vrClass instance of lower level.

In Figure 4, the higher level vrClass instance Student effectively acts as a specialisation of a lower level vrClass instance Person. Since Person is the only key component for Student, it determines its identity, so that each Person object is a Student object (shares the identity with it), while conversely not all Person objects are also Student objects. It should be noted that the participation constraint for Student in Person given in the figure is redundant, as it is implied by the primary key dependency for Student.

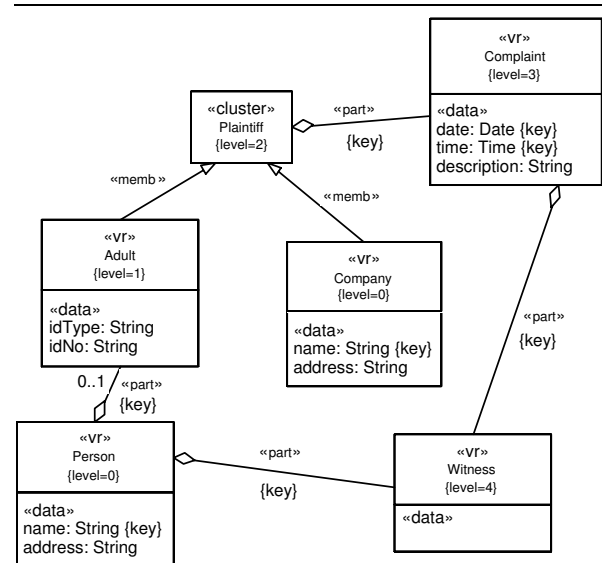


Figure 5. A more full-featured HERM diagram example.

Finally, in Figure 5 we demonstrate a more complex

HERM diagram, featuring 5 levels of Component instances, including a Cluster instance. It also demonstrates the usage of higher level vrClass instances in a variety of ways (as a “relationship” in case of Witness, as an aggregate part in case of Complaint, and as a “specialisation” in case of Adult).

To complete the discussion of the UML profile for HERM introduced in this section, we observe that for as long as a model uses only the stereotyped model elements in a syntactically correct way, the significant body of HERM models completed so far for various applications serves as a guarantee that the model has a precise semantics and is consistent in the sense that it is value-representable and that an equivalent relational model exists and can be constructed by using a well-understood algorithm with no loss of semantics. In fact, the semantics for HERM is defined more formally than for UML meta-models like UML class diagrams since, for the sake of flexibility and backward compatibility, the UML specification does not apply formal methods to specify semantics.

On the other hand, there is no need to limit oneself to the features explicitly defined in the profile. It would be (from the point of view of the UML specification) syntactically correct to combine the use of value-representable classes with features such as operations, visibility etc., that stereotyped classes inherit from UML classes (at meta level). Also, it should be possible to use attributes not stereotyped as «data» and to use «vr» classes in the same diagram as conventional UML classes. In fact, such a usage may be advantageous for example in cases when some classes need to be persisted to a relational database (and hence, should be mapped to a relational model), while others are used to model compartmentalisation of application logic, and for them value-representability is not desired. However, in case of such extended usage, the semantics of models need to be constantly carefully re-examined, as mathematical foundation is not necessarily available.

4. Conclusion

We use the Object Management Group (OMG) Meta Object Facility (MOF) to formally describe the Higher Order Entity Relationship (HERM) model. In addition, we formally define a UML profile for HERM. By demonstrating that MOF and UML can represent the most critical capabilities of HERM, we open way for better integration of HERM database models in projects relying on UML for application modelling and for the use of UML tools for HERM modelling.

References

- [1] B. Daum. Asset oriented modeling (aom) tool support, 2004. <http://www.aomodeling.org/tools.htm>.
- [2] A. Hall. Seven myths of formal methods. *IEEE Software*, 7:11–19, 1990.
- [3] E. Marcos, B. Vela, and J. M. Cavero. A methodological approach for object-relational database design using uml. *Journal on Software and Systems Modeling*, 2:59–72, 2003.
- [4] R. J. Muller. *Database Design for Smarties: Using UML for Data Modeling*. Morgan Kaufmann, 1999.
- [5] E. J. Naiburg and R. A. Maksimchuk. *UML for Database Design*. Addison-Wesley, 2001.
- [6] OMG. Meta object facility specification, 2002. <http://www.uml.org>.
- [7] OMG. UML 1.5 specification, 2003. <http://www.uml.org>.
- [8] OMG. UML 2.0 superstructure specification, 2004. <http://www.uml.org>.
- [9] K. D. Schewe, J. W. Schmidt, and I. Wetzel. Identification, genericity and consistency in object-oriented databases. *Lecture Notes in Computer Science*, 646:341–356, 1992.
- [10] B. Thalheim. *Entity Relationship Modeling*. Springer, 2000.