Enhancing Identification Mechanisms in UML Class Diagrams with Meaningful Keys

C. Maria Keet
School of Computer Science
University of KwaZulu-Natal
Durban, South Africa
keet@ukzn.ac.za

ABSTRACT

Unlike identification with keys and reference schemes in ER and ORM, UML uses internal, system-generated, identifiers, with a little-known underspecified option for user-defined identifiers. To increase the ontological foundations of UML, we propose two language enhancements for UML, being formally defined *simple* and *compound identifiers* and the notion of *defined class*, which also have a corresponding extension of UML's metamodel.

Categories and Subject Descriptors

I.2.4 [**Knowledge Representation Formalisms and Methods**]: Representation languages; H.1.m [**Information Systems**]: Models and Principles—*Miscellaneous* ; H.2 [**Database Management**]: Data Models

Keywords

Identity, Identification, Key, Defined Class, Quality of models and their languages

1. INTRODUCTION

In conceptual data modelling literature, identification and identity are often used interchangeably and despite the long history on the topic, there is a remarkable lack of agreement about what they are, how identification mechanisms—keys, reference schemes, identifiers—are dealt with in graphical languages such as UML and EER, how to formalise it, and how modelling tools implement them. Ascertaining when a particular key is a good one has been investigated before, and the reader is referred to [7, 12] for lucid examples of good and bad keys. However, these and other works on identity and identification in conceptual data modelling do not address the logical and ontological underpinnings of identity and identification with keys, i.e., whether the identification mechanism itself is a good one or not. In addition, one needs to decide whether the *procedure* to find and represent identity, or at least good keys, should be (*i*) be a step in the modelling methodology; (*ii*) also be enforced in the CASE tool; or (*iii*) be part of the metamodel and/or in the conceptual modelling language itself, and whether that should be the same for all conceptual data modelling languages. A uniform, or at least a structured and unambiguous approach, can reduce or even avoid inconsistencies in a conceptual data model and achieve interoperability through less resource-consuming information integration. The present lack of harmonisation in handling identity and identification hampers this.

More specifically, the UML v2.3 specification [11] contains many occurrences of the terms “identity”, “identification”, and “identifier”, but does not provide detail on how exactly the identification system is implemented, other than using the *internal identifier* mechanism where the identifiers are generated and assigned by the system. This poses challenges both for devising subclass hierarchies—a subclass with a semantically incompatible identity is incorrect with respect to the information, but remains erroneously in the model without further machinery to detect it—and for application integration, because it requires a manual re-assessment of the information and each data point to elucidate which objects are the same. Concerning the latter, the recent data integration problems for the Johannesburg area integrated services delivery information management system may serve as an example of the pressing need to resolve this; at present, there are no tools that do this automatically. Concerning the former, take, e.g., Social entity (e.g., the SABC) as a subclass of Group of people: instances of the former can change its members whilst being the same object, but the latter cannot, hence, they have different identity criteria, which thus will cause problems in the OO system if implemented as such nevertheless. This and similiar problems have been addressed for ontologies with the OntoClean method [5], but there is no usable counterpart in conceptual data modelling regarding the detection of such incompatible identity criteria, because there is no unambiguous way to deal with them.

Ontology-driven information systems and conceptual data modelling are gaining momentum [3, 4, 6, 9], which demonstrate that reusing notions from Ontology leads to better quality conceptual data models. Here we take a similar approach aiming at addressing the issue of identification in UML Class Diagrams so that the subject domain semantics can be represented more comprehensively and thereby foster the development of better quality UML Class Diagrams that can generate both better software systems and that can be more easily aligned or integrated with other information systems. Due to space limitations, we only summarise the outcome of the ontological and logical analysis of identity and identification mechanisms; the motivations are explained and discussed in the extended version of this paper [8]. Those more detailed insights are used here to propose two language enhancements for UML Class Diagrams, being formally defined *simple* and *compound identifiers* and the notion of *defined class*, which also have a corresponding extension of UML's metamodel.

We summarise identification mechanisms in conceptual data modelling in Section 2. Extensions are proposed in Section 3, and we close in Section 4.

2. IDENTIFICATION IN EER, ORM, AND UML

Keys are incorporated in conceptual models in different ways, which is summarised for simple and complex keys. This enables one to refine the identification mechanism in UML and harmonise them with ER/EER, Object-Role Modelling (ORM), and the Web Ontology Language (OWL).

ORM is most explicit in its use of an identification mechanism, which is also enforced through diagram validation in the two well-known ORM CASE tools VisioModeler 3.1 and NORMA. Each entity type has a so-called *reference scheme*, which in its simplest form—then dubbed *reference mode*—is a binary relationship from that population to the values of some value type [7], with a *mandatory* participation by the entity type and a *1:1 uniqueness* constraint. Let us formalise this as follows:

*Definition 1. (ORM reference mode (*REFM*)) An ORM* reference mode *for an object type is a identification mechanism that provides a sufficient condition for identity such that: let C be the object type, V a value type, I relating C and V, i.e., $\forall x, y(I(x, y) \rightarrow C(x) \wedge V(y))$, and constrained to $\forall x(C(x) \rightarrow \exists y I(x, y))$, $\forall x, y, z(I(x, y) \wedge I(x, z) \rightarrow y = z)$, and $\forall x, y, z(I(x, y) \wedge I(z, y) \rightarrow x = z)$.*

The single-attribute primary key, *simple key* (SKEY), in ER and EER has essentially the same definition as REFM but without the mandatory constraint [2].

ORM and EER allow more advanced identification than just REFM and SKEY. These include n-ary keys with $n > 1$, weak entity types, and external uniqueness. EER's n-ary key that spans more than one attribute (i.e., for an entity type with n attributes, $n \geq 2$, then the key spans at most n attributes) can be formalised as follows:

*Definition 2. (Multi-attribute key (*MKEY*)) An ER/EER* multi-attribute key *for an entity type is a identification mechanism that provides at least a sufficient condition for identity such that: let C be the entity type, $V_1, \dots V_n$ data types, I relating C and $V_1, \dots, V_n$, i.e., $\forall x, y_1 \dots y_n(I(x, y_1, \dots, y_n) \rightarrow C(x) \wedge V_1(y_1) \wedge \dots \wedge V_n(y_n))$, and constrained to $\forall x(C(x) \rightarrow \exists y_1, \dots, y_n I(x, y_1, \dots, y_n))$ and for each V_i, with $2 \leq i \leq n$, denoted together with $\bar{y}$ that participate in the key (and with $\underline{y}$ those attributes that do not), $\forall x_1, x_2, \bar{y}, \underline{y}(I(x_1, \bar{y}, \underline{y}) \wedge I(x_2, \bar{y}, \underline{y}) \rightarrow x_1 = x_2)$ and $\forall x, \bar{y}, \underline{y}, \bar{z}, \underline{z}(I(x, \bar{y}, \underline{y}) \wedge I(x, \bar{z}, \underline{z}) \rightarrow \bar{y} = \bar{z})$.*

With a minor variation in the definition ($1 \leq i \leq n$ instead of $2 \leq i \leq n$), one can generalise to an 'EER key' that subsumes both MKEY and SKEY. ORM's analogue of MKEY is referred to as an internal uniqueness constraint that spans > 1 ORM roles where the other types are not constrained to "$V_1, \dots V_n$" but may also be object types. These type of keys are also referred to as *natural keys* for they seek to use real attributes of the objects to identify them, compared to artificial keys like student matriculation numbers.

The ontological status and formalisation of ER/EER's weak entity type and ORM's external uniqueness with a compound reference scheme is less clear. Let us take ORM's approach [7] and place this in a more comprehensive and precise definition that also respects ORM and EER CASE tool implementations:

*Definition 3. (Compound reference scheme (*CREFS*)) A* compound reference scheme *is an identification mechanism that provides a sufficient condition for identity, where an object type C participates in m roles ($m \geq 2$) in m separate binary relationships $I_1, \dots, I_m$, the range of each I_i is either C_i or V_i, and for each i (with $1 \leq i \leq m$), $\forall x(C(x) \rightarrow \exists y I_i(x, y))$ and $\forall x, y, z(I_i(x, y) \wedge I_i(x, z) \rightarrow y = z)$, and $\forall x_1, x_2, y_1 \dots y_m(I_1(x_1, y_1) \wedge \dots \wedge I_m(x_1, y_m) \wedge I_1(x_2, y_1) \wedge \dots \wedge I_m(x_2, y_m) \rightarrow x_1 = x_2)$.*

ER's weak entity type [2] is the same as CREFS, including the mandatory participation. Thus, the only thing that has changed compared to REFM is the *amount* of relationships involved, but the principle remains the same. Put differently, REFM is a special case of CREFS.

UML uses the *internal identifier* mechanism where the identifiers are assigned by the system, of which its problems are discussed from an implementation perspective in [12]. One neither can specify manually if that should occur under the constraints of REFM or SKEY. Halpin and Morgan [7] propose to insert a user-defined identification by appending the preferred attribute with "`{P}`", which amounts to an *alternative* identifier that bears some subject domain semantics compared to the internal identifier, i.e., in the direction of a natural key, and slightly closer to the notion of identity in ontology. The UML Ontology Definition Metamodel (ODM) v1.0's proposal to handle ER keys [10] is depicted in Fig. 1-A, and has an optional UML package where Property is extended with Identifier and where alt:string is the "name of instance of alternative identifier if there are more than one identifiers for a given class" (Fig. 1-B). Problems with this proposal are that it does not specify multiplicity constraints on the identifier and it does not distinguish between REFM, MKEY, and CREFS.

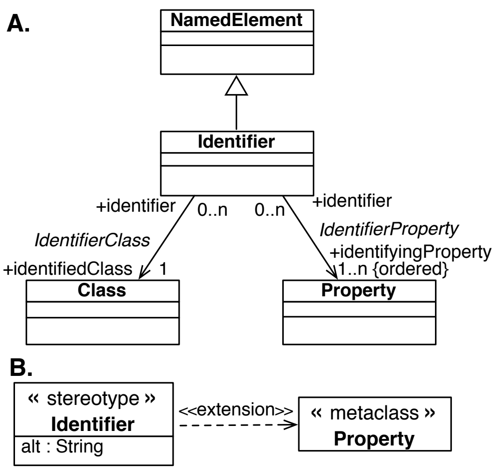

Figure 1: A: OMG's extension for keys/identifiers (p291); B: its UML profile (p292) [10].

3. EXTENSIONS TO UML

To put the preceding analysis to use, we first address inclusion of a more precise and comprehensive way of identification in UML Class Diagrams, using ER keys and ORM's reference schemes—i.e., REFM, MKEY, and CREFS—as a basis, and subsequently examine the feasibility of adding OWL's notion of 'defined class' from ontologies into UML to assert that, with respect to the subject domain, a particular combination of properties fully define the class, hence, that the combination of properties determine the class and vv.

3.1 UML's identifiers

Due to the lack of a formal specification of UML's identifier, anyone can interpret the text on UML keys the the ODM [10] (pp. 290-293) to one's liking, which we aim to do here in the light of the identification schemes in (E)ER and ORM and the ontological guidance described in [8]. ODM's intention is to capture all three types of (E)ER keys (simple and multi-attribute key, and weak entity type) in the one alternate key extension as depicted in Fig. 1-A. Although ER does not have mandatory participation in the keys as introduced by Chen [2], because there unidentified objects were allowed, this does not hold for UML, because it is subsumed by NamedEntity and uses the internal identifiers; hence, ORM's REFM and CREFS are applicable. This entails that, first, the order of the properties of the identifier do not matter, hence {ordered} can be omitted. Second, we have to add that the property has a multiplicity of exactly one, which can be done by setting the Multiplicity option of Property to 1; this enforces the default value, yet it also prevents modifications, so that a common meaning of the identifier is ensured. Because of this change, which clearly does not hold for all properties, a new subclass is introduced, IdentifyingProperty, which must participate in at least one Identifier. It still has to incorporate UML's distinction between attribute of a class and association between classes. The ODM restricts Property to association end or attribute of the class or its superclass for identifiers [10] (p292), so that one can distinguish between an external identifier that involves at least one identifying association end and possibly one or more attributes, yet keep the option to also have only identification within the class (i.e., an identifier composed of only attributes). These refinements are depicted in Fig. 2 (note: ExternalIdentifier's associations have only the multiplicity drawn to avoid cluttering the diagram, but the same adornments apply as with IdentifierProperty).

Thus, we have captured more precisely UML's description and intention of inclusion of keys with CREFS as a basic formal foundation and matching extension to the ODM as depicted in Fig. 2. Note that this can be represented easily also in the $\mathcal{DLR}_{ifd}$ DL language that Berardi et al. used to formalise a large fragment of UML Class Diagrams [1], principally thanks to the **id** construct that was introduced specifically to handle complex keys and weak entity types.

3.2 Defined classes in UML

From the representation of natural keys with CREFS and Fig. 2, it is a small step to also include the notion of defined class, in analogy to defined class in OWL, which, loosely, amounts to identifying—more precisely, providing the mandatory and sufficient conditions—by a set of one or more properties. In addition to more detailed conceptual modelling by representing the subject domain semantics more precisely, one also may gain benefits with automated reasoning

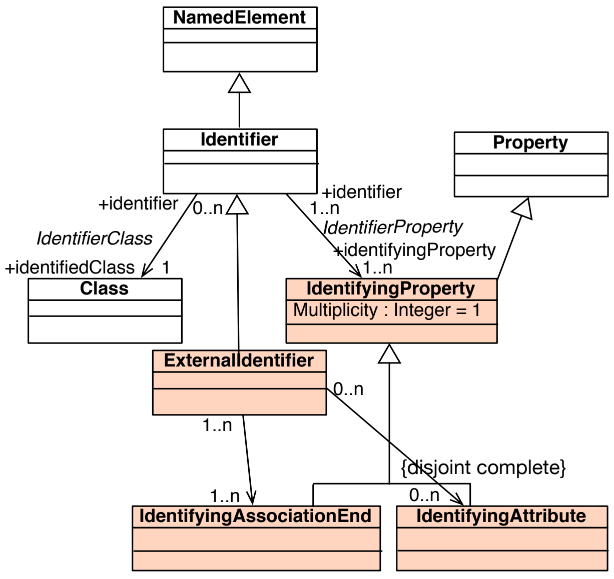

Figure 2: Extension to the UML ODM's identifier (only multiplicity is drawn for ExternalIdentifier), covering REFM, MKEY, and CREFS.

over formal conceptual data models. To declare that certain properties amount to the combination of at least mandatory (and possibly necessary) *and* sufficient conditions in the ontological sense, one can use a formal definition, add it in some way to UML's metamodel, or exploit OCL. The former is trivial: defining Defined Class is straightforward with "≡" or `EquivalentClass` (in OWL) with their semantics, so that with the to-be-defined class on the left-hand side, the right-hand side of the definition can have any property. Regarding UML's OCL, this might work, but it does not pose any restrictions on how to use it consistently and there is no conveniently usable graphical element for it. A possible extension to the UML metamodel by subtyping Class and Property is shown in Fig. 3-A, which is the configuration with the least amount of objections and restrictions. It limits any class stereotyped as «Defined class» to be composed of attributes and/or association (ends). IdentifyingProperty's attribute isReadOnly (inherited from Property) is set to true, which guarantees it is not changeable, hence, its subclasses cannot override this property with some conflicting attribute declaration. Also, isDerived is set to false, because one's identity should not be dependent on implicit knowledge in the conceptual data model (if it would be derived, then one has to use the original classes/attributes for identification instead). Last, the isAbstract attribute of Class is by default set to false already because abstract classes are not assumed to be instantiated directly anyway—hence do not face the issue of identity or identification in the information system—but its subclasses denoting some piece of reality or other subject domain directly, are.

Fig. 3-B demonstrates an example with MedalWinner's properties that make up the identifier, and the identifier inside of Competition consisting of three attributes, i.e., a CREFS and a MKEY, respectively. The "«identifier alt=a»" follows through the initial proposal by [10], where the "a" is a chosen string to denote which properties belong together to make up a key,

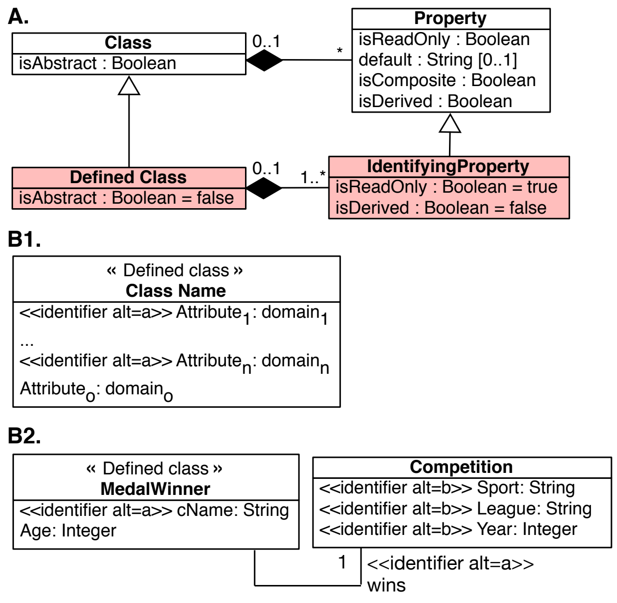
A.
Class
isAbstract : Boolean
0..1
*
Property
isReadOnly : Boolean
default : String [0..1]
isComposite : Boolean
isDerived : Boolean
Defined Class
isAbstract : Boolean = false
0..1
1..*
IdentifyingProperty
isReadOnly : Boolean = true
isDerived : Boolean = false
B1.
« Defined class »
Class Name
<<identifier alt=a>> Attribute1 : domain1
...
<<identifier alt=a>> Attributen: domainn
Attributeo: domaino
B2.
« Defined class »
MedalWinner
<<identifier alt=a>> cName: String
Age: Integer
Competition
<<identifier alt=b>> Sport: String
<<identifier alt=b>> League: String
<<identifier alt=b>> Year: Integer
1
<<identifier alt=a>>
wins

Figure 3: A: Extension (shaded classes) to UML's metamodel with Defined class and IdentifyingProperty. B: Examples of defined classes with n mandatory and sufficient conditions and one other attribute, and a refinement of OMG's Olympics example [10].

as shown in Fig. 1-B. Thus, this mechanism also trivially caters for ER's and ORM's alternate keys.

3.3 Discussion

With UML's metamodel, IdentifyingProperty inherits the attributes isReadOnly and isDerived from Property and Defined class inherits isAbstract, so they *have to* be set to some value to ensure consistency in meaning. Also, IdentifyingProperty stands for 'property used in identification' and its constraints concern *how to use it uniformly* in UML Class Diagrams; it definitely does *not* make an ontological commitment as to *the possibly different nature of* properties of identity. That is, the proposed extensions are focussed on practical usability in conceptual data modelling, informed by ontology. The proposals for conceptual data modelling with UML Class Diagrams are usable approximations to the qualitative, relative, and synchronic identity, and to the notion of OWL's and DL's defined concept discussed in [8].

Although UML's ease of system-generated identifiers relieves the burden of detailed conceptual analysis by the modeller, it is exactly making implicit subject domain semantics explicit that is crucial in the analysis stage; or: less analysis during the modelling stage stores up more problems down the road in terms of software bugs and interoperability. Therefore, even if one would to want to keep internal identifiers, then at least the option to describe a user-defined alternate key should be encouraged in the tools and recommended to be used. This can be done with the proposed refinements, which correspond to SKEY, REFM, and MKEY of ER/EER and ORM and Fig. 2, i.e., the user-defined natural keys or *semantic identifiers*. One can then expand on it with the notion of defined classes, which is already possible with the logical counterpart of the languages, and now also more precisely as proposed in Section 3.2 and illustrated in Fig. 3. The adornments of the graphical language is in line with existing practices in UML, i.e. the stereotyping with «Defined class», «identifier», and «identifier alt=a».

Having a more precise way of identification in UML is the first step. To put it to good use, modelling methodologies have yet to be developed to extract good attributes and associations from the subject domain, and additional rules have to be devised for cleaning up and checking ontological classification of a taxonomy in UML Class Diagrams, alike OntoClean for ontologies. Also, mapping assertions between ER/EER, ORM and UML diagrams are yet to be implemented to facilitate information integration.

4. CONCLUSIONS

UML uses internal identifiers that do not bear any meaning, whereas ER, EER, and ORM aim at natural keys that do emphasize the properties of the entity type, which was formalised unambiguously. To increase the amount of ontological foundations of UML, we have proposed two language enhancements for UML, being formally defined *simple* and *compound identifiers* and *defined classes*, which also have a corresponding extension of UML's metamodel.

5. REFERENCES

[1] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1-2):70–118, 2005.

[2] P. P. Chen. The entity-relationship model—toward a unified view of data. *ACM TODS*, 1(1):9–36, 1976.

[3] N. Guarino. Formal ontology and information systems. In *Proc. of FOIS'98*. Amsterdam: IOS Press, 1998.

[4] N. Guarino and G. Guizzardi. In the defense of ontological foundations for conceptual modeling. *Scandinavian J. of Info. Sys.*, 18(1):9p, 2006.

[5] N. Guarino and C. Welty. An overview of OntoClean. *Handbook on ontologies*, pages 151–159. Springer, 2004.

[6] G. Guizzardi. *Ontological Foundations for Structural Conceptual Models*. Phd thesis, University of Twente, The Netherlands. Telematica Instituut Fundamental Research Series No. 15, 2005.

[7] T. Halpin and T. Morgan. *Information modeling and relational databases*. Morgan Kaufmann, 2nd ed., 2008.

[8] C. M. Keet. Enhancing identification mechanisms in UML class diagrams with meaningful keys (extended version). Technical Report SoCS11-1, School of Computer Science, University of KwaZulu-Natal, August 2011.

[9] C. M. Keet and A. Artale. Representing and reasoning over a taxonomy of part-whole relations. *Applied Ontology*, 3(1-2):91–110, 2008.

[10] Object Management Group. Ontology definition metamodel v1.0. Technical Report formal/2009-05-01, Object Management Group, 2009. `http://www.omg.org/spec/ODM/1.0`.

[11] Object Management Group. Superstructure specification. Standard 2.3, Object Management Group, May 2010. `http://www.omg.org/spec/UML/2.3/`.

[12] R. Wieringa and W. de Jonge. Object identifiers, keys, and surrogates—object identifiers revisited. *Theory and Practice of Object Systems*, 1(2):101–114, 1995.