THE USE OF UML CLASS DIAGRAMS TO TEACH DATABASE MODELLING AND DATABASE DESIGN

Bernadette Marie Byrne
School of Computer Science
University of Hertfordshire
College Lane
Hatfield
AL10 9AB
b.m.byrne@herts.ac.uk

Yasser Shahzad Qureshi
School of Computer Science
University of Hertfordshire
College Lane
Hatfield
AL10 9AB
Yasser.Shahzad@gmail.com

ABSTRACT

It is now nearly 30 years since Peter Chen's watershed paper "The Entity-Relationship Model – towards a Unified View of Data". [1] The entity relationship model and variations and extensions to it have been taught in colleges and universities for many years. In his original paper Peter Chen looked at converting his new ER model to the then existing data structure diagrams for the Network model. In recent years there has been a tendency to use a Unified Modelling Language (UML) class diagram for conceptual modelling for relational databases, and several popular course text books use UML notation to some degree [2] [3]. This paper looks at the usefulness of using UML class diagrams for teaching database design in undergraduate courses. In this paper we look specifically at two concepts which can cause problems for the novice database designer. Firstly transferring the concept of a weak entity from an Entity Relationship model to UML and secondly the notation for structural constraints in different diagramming notations. We also look at the mixture of notations which students mistakenly use when modelling. This is often the result of different notations being used on different courses throughout their degree. Peter Chen wrote in his original paper "The entity-relationship model can be used as a tool in the structured design of databases using the network model" today we could write "the UML class diagram can be used as a tool in the structured design of databases using the relational model". Or can we?

Keywords
EERD, UML Class Diagram, Relational Database Design, Structural Constraints

1. INTRODUCTION

The ER model was originally put forward by Chen [1] and subsequently extensions have been added to add further semantics to the original model; mainly the concepts of specialisation, generalisation and aggregation. In this paper we refer to an Entity-Relationship model (ER) as the basic model and an extended or enhanced entity-relationship model (EER) as a model which includes the extra concepts. The ER and EER models are also often used to aid communication between the designer and the user at the requirements analysis stage. In this paper when we use the term "conceptual model" we mean a model that is not implementation specific. Past work has been on-going to investigate the usability and quality of ER and EER models [4], [5], [6], [7], [8], [9], [10], [11], [12] and [13].

2. COMPARISONS BETWEEN A UML CLASS DIAGRAM AND AN ER DIAGRAM.

Dr. Peter Chen proposed the entity relationship model to present a *unified view of data*, in the same way as UML attempts to present a *unified modelling language*. In this paper we assume the reader is familiar with EER and UML class diagrams but for completeness we present a short comparison here.

The Unified Modelling Language (UML) is a standardised modelling language and was proposed to support and extend the Object-Oriented Analysis and Design methods that appeared in the early '90s, [14]. UML was initially developed at Rational Software and then later on was adopted by the Object Management Group (OMG). Since then the OMG has been managing UML. UML focuses on OOAD (Object Oriented Analysis and Design), and therefore has proposed a range of diagrams to provide support for these purposes. UML contains quite numerous diagrams for designing software systems. For the purposes of this paper we are only going to focus on Class Diagrams out of the UML collection of diagrams.

The essence of class diagrams is to help us model the real world in terms of objects and their relations. Class diagrams can be used to describe the entities, data, internal class structure, class inter-relationships and associations between two or more classes in order to determine the static structure of a system [15].

Blaha, M [16], describes the UML Class diagram as a dialect of Chen's original ER Diagram. Both of the notations are used to depict real world objects in terms of a static view of the system. One of the reasons, which is unanimously accepted, for the use of UML in database design is, the advent of object relational and object oriented databases.   However, Relational Database Management Systems were inherently not meant to be object oriented as they depict information in the forms of two-dimensional tables consisting of rows and columns forming individual cells to store atomic units of data.

2.1 *Relationships* and Associations in EERD and UML Class Diagrams

2.1.1 Class v Entity

Classes are the most important building block of any object-oriented system. Normally defined as a template of a set of objects that share same attributes, have the same operations and the same relationships with other classes. Classes can be used in the same manner as entities in a way that anything of interest/significance to the system can be defined as a class whether it is a software "thing", hardware "thing" or even a "conceptual thing", [17]. The diagrammatic notation is a rectangle box divided into sections for Class Name at the top and then its attributes in the middle section and operation in the last section.

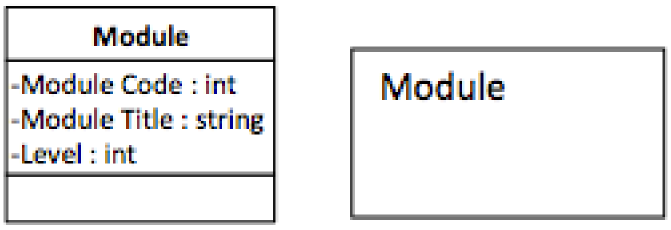

Figure 1 – UML Class and ER Entity notations

An entity is presented in much the same way. Although the ER model being inherently a conceptual model would not initially show attributes. However at later stages of abstraction attributes can be added to the model as in the notation of Elmasri and Navathe [2]. Indeed when using a CASE tool to draw an Entity Relationship diagram there is often the option to either switch-on or switch off the attributes on the Model.

2.2 Attributes in ER and in Class diagrams

An attribute on a UML class diagram, is similar to the attributes in an ERD. Attributes of a class describe the information that a class contains, also associated with attributes is the domain of the values. Attributes are described to be as a property of a thing (class). Attributes are normally depicted as plain text in the class rectangle box as in Figure 1.

2.3 Operations:

An operation is a behaviour that an object or class can perform. In other words an operation is an abstraction of something that the objects of a class do and is shared between all the objects. You may have numerous operations in a class or none, [17]. Operations are also defined in the same box that is used for the class and are defined as programing functions using a parenthesis bracket. There is no equivalent of this on an ER Diagram.

2.4 *Cardinalities/Structural Constraints/Multiplicities:*

In essence both modelling techniques have the same levels of support for showing cardinalities and mandatory and optional participation between two entities/classes - with one major difference, UML uses “look across” notation and ER, on the whole, uses “look here”. This is explained in the following familiar *employee works for department* example.

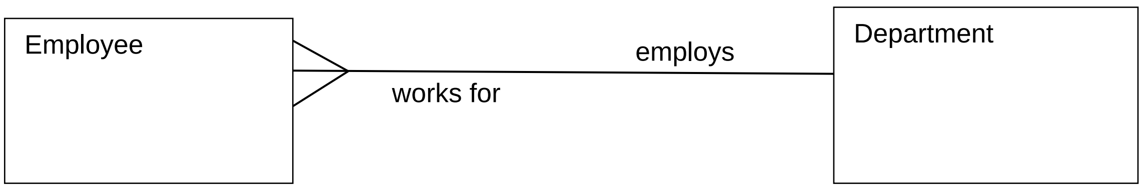

Figure 2

A cardinality constraint shows the maximum number of entity instances that can occur in the relationship. In this case a department employs many employees and we are looking across at the crows feet on the adjoining entity to read the cardinality. Likewise we look across from the employee entity to see that an employee only works for one department.

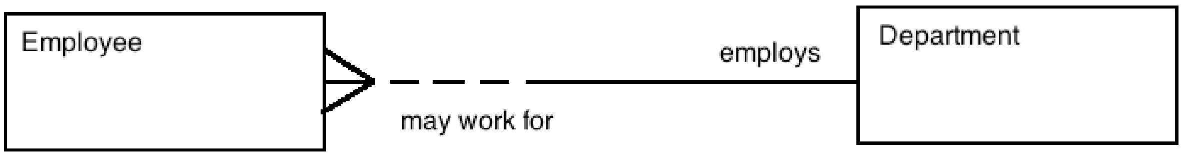

Figure 3

If we change the situation and now not all employees have to work for a department and therefore the relationship “Employee works for Department” becomes optional, we could insert a broken line on the employee side of the relationship line as in Figure 3 above. Now we are still looking across for the cardinality but “looking here” for the participation.

Some notations use a minimum and maximum number beside the entities to denote both cardinality and participation. UML uses minimum and maximum numbers but with a “look across” notation. EER Elmasri and Navathe notation uses minimum and maximum numbers but with a “look here notation”.

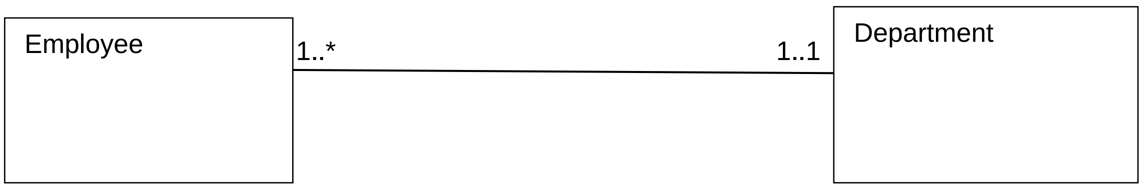

Figure 4 UML example. Look across notation – mandatory relationships

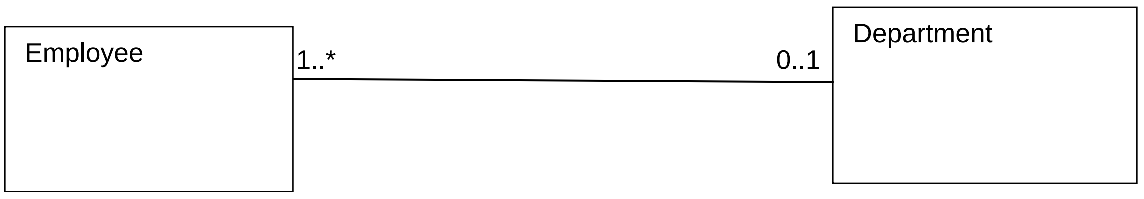

Figure 5 UML example Look across notation – optional for an Employee to work for a department.

In the examples in figure 4 and 5 the minimum number is the first number and represents the minimum number of times an entity instance can take part in the relationship. Therefore if the first number is one this denotes that the relationship is mandatory. If the first (minimum) number is zero (0) this denotes optionality. The second (Maximum) number is the maximum number of times an entity instance can take part in the relationship. If it is many we can use a *.

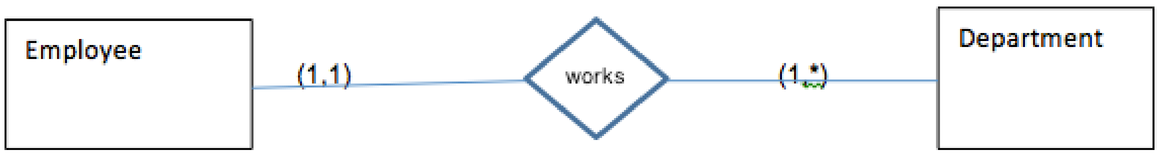

Figure 6 Elmasri and Navathe notation – Look here notation.

The various combinations of "look across" and "look here" that occurred in ERD's in 1995 are well documented in Song's paper [10]. As we can see from Figure 3 we can also have a combination such as "look here" and "look across" in the same set of constraints.

2.5 Association v Relationship

Association is a relationship between two classes to determine how they interact with each other. The association can be, in general, classified as a relationship between two or more classes. Associations can further be split into specific types of relations that might exist between classes given in a particular system. This relation is displayed as a simple line connecting from one class to the other annotated with an appropriate name for the role of the class. We can compare this to a relationship link between two entities on an ER diagram.

Both UML and EER can show specialisation and generalisation and the expected constraints. This is a straightforward conversion from one model to the other. We will deal with this in a further paper. Also both show m:n relationships which we do not deal with here.

2.5.1 Aggregation/Composition

Before we explain the concepts of aggregation and composition, which actually are a more expounded form of a "Part Of" relationship between classes/entities, we need to understand the "part-of" relationship.

Part-of Relation: A class may be a part of another class, and play a whole-part relation with another class. While doing so a class may play an immutable role, or its existence may not be dependent upon another class's existence. Suppose we say Thumb is part of a Person and Person is a part of Company, but then it does not mean that thumb is part of a Company. Hence, as such there is no relationship between a company and the thumb object. [21]

In UML aggregation and composition are kinds of associations as they both are ways of recording that an object of one class is part of an object of another class.

A simple definition of Aggregation is that it is a special form of association where by an element/entity type contains other element/entity type. It indicates a "whole-part" relationship between two entity types. Aggregation is an abstraction concept of building objects, which are composite, from their component objects. There can be different types of aggregation i.e. "IS-A-PART-OF", "IS-A-COMPONENT-OF". Normally phrases like "consists of", "contains", "is part of" etc. identify aggregation relationships between classes/entity types.

One example of aggregation could be that a car consists of wheels, an engine, a chassis, a gearbox and so on. Car itself is an object made up of other objects. Furthermore an engine is composed of many other objects such as pistons etc. The pistons that compose the engine will eventually wear out i.e. pistons will be destroyed along with the engine, as soon as the engine ends its lifetime, thus the relationship between the two will be considered as that of composition aggregation.

A composition aggregation is a special kind of aggregation in which the class containing the object of another class owns the containing object; it has a stronger ownership. The parts live inside the whole object and will be destroyed with the whole object. Hence, composition is aggregation with further restrictions imposed on the relationship in terms of ownership. Composition can be defined as a strong form of aggregation with a lifetime dependency between each part, and the whole. No part can belong to more than one composition whole at a time, and if the composite whole is deleted its parts are deleted with it.

Aggregation is a mechanism for forming a whole from components and parts. It is a special type of association in which objects are assembled together to create a more complex object. Aggregation provides a means to display a whole-part relationship between classes where it exists. In such a relationship one class contains the object of another class and has access to all the functionality of that class through the contained object.

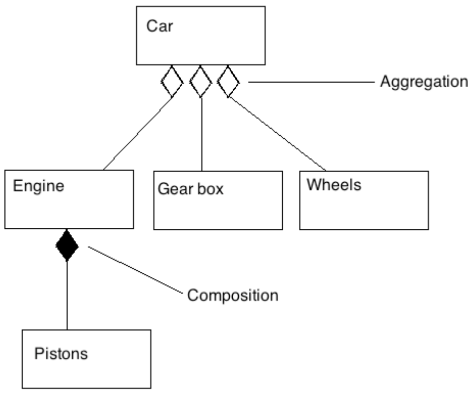

Figure 7 UML notation for Aggregation and Composition.

In ER, aggregation is the same concept as in UML. In ER aggregation can be defined as whole-part relationship between entity types. In ER primarily the focus is to aggregate entities that are logically related to each other in a way that they complement one another to form the whole. So in essence aggregation in ER is a way to model a relationship between a collection of entities and relationships.

2.5.2 Aggregation in EERD and UML Class Diagram

Both UML Class Diagrams and EERDs support aggregation; the only difference is in the way it is depicted in both modelling techniques.

Aggregation in EER is quite simple as it is at more of a conceptual level and simply encapsulates the aggregated entities in one cloud like figure showing that the entities are in a "whole part" relation. It is vital to keep in mind that in the OO paradigm, aggregation is mostly referred to as a "Has a" relation but has the ability to depict a "whole part" relation between classes using two different notations and concepts of aggregation and composition. Though different authors have used the terms "has a" and "whole part" interchangeably, adding to the misunderstanding that already exists between the two concepts but the difference, that we make here, between the two is that we perceive "Has a" relation as a composition; where one entity is amalgamated into the other and cannot exist without the other, whereas; "Whole part" relation is more of a situation where one of the entities complements the other, but still is separate in terms of its existence and can exist itself on own.

The point to be highlighted here is that though the term is the same i.e. "whole part" but it still has a slightly different meaning for both the OO paradigm and the relational paradigm. As relational database do not support tuples to be stored as attribute values, amalgamating one entity within the other is not an option in relational databases. But, at the conceptual level it is considered vital to depict the "whole part" relation between entities to help database administrators understand the enterprise level complexities which exist in a database.

The main reason for confusion between the two models and their support for aggregation, that we comprehend, is that both the models have notations for aggregation but the capabilities of the OO paradigm and the relational paradigm at the implementation level are just not the same. The OO paradigm is far more capable than relational databases at low level implementation for aggregation and composition, whereas relational database consisting of tables, rows and columns just simply cannot support the aggregation to the same extent as most OO programing languages do. Programing languages have the ability to manipulate information in volatile memory whereas relational databases have to store the information on a hard drive i.e. physical memory. This highlights the differences between their capabilities which then subsequently have impacts on the modelling techniques presented for both the OO paradigm and the relational paradigm i.e. UML Class Diagrams and EERDs respectively.

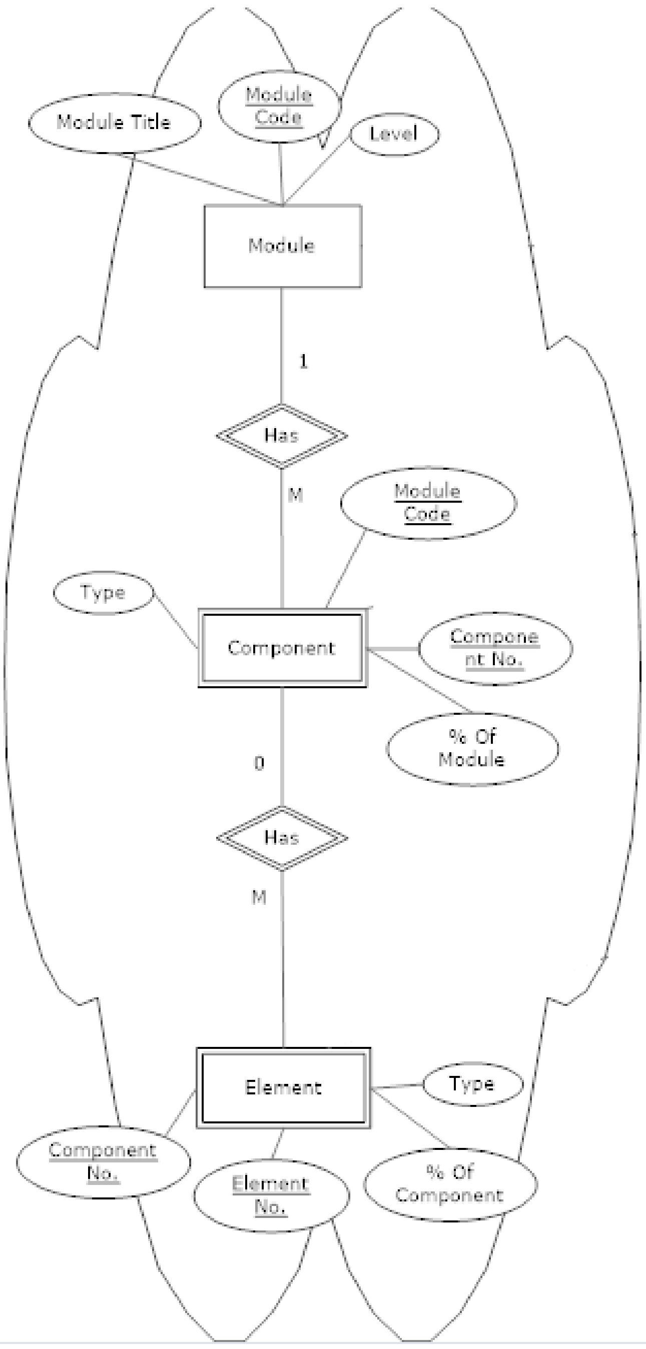

Figure 8 ER notation for aggregation (Module may have many components, components may have many elements)

The problem with using UML Class Diagrams as a modelling technique for relational databases is that a programmer and a database administrator may not perceive them in the same manner. At a first glance a programmer, who deals with classes and code all day long, may perceive it as a normal class diagram. We must consider how people have different mind sets and how their psyche influences the use of such techniques. Out of intuition a programmer will have a different perception of the same UML Class Diagram than that of a database administrator.

3. PROBLEMS FOR NOVICE DATABASE DESIGNERS

We will now examine two problems encountered by novice database designers when transferring from an ER diagram to a UML class diagram. The first problem is converting from the ER representation of a weak entity to an equivalent UML representation. The second problem is having different positions for the structural constraints on the diagrams.

3.1 Weak Entities

Weak Entities are a common modeling construct in conceptual data modeling and were described in Peter Chen's original [1] paper on entity-relationship modeling. A weak entity is described as when an entity in a binary relationship has identification dependency and existence dependency on the 'parent' entity in the relationship. The primary key of the weak entity is made up of the combination of the primary key from the parent entity and a partial key of the weak entity. The existence of the weak entity is often said to be dependent on the parent entity. Previous work by others in this field has argued that weak entities belong to an intermediate design stage and should not be present in the final design [18], or that weak entities should be used with caution as they have no theoretical basis [19]. However, we take the point of view that the weak entity structure is ubiquitous in conceptual modeling and in order to provide semantic clarity the weak entity structure needs to be clearly represented in modeling methods.

Song et al [10] in 1995 carried out a comprehensive comparative analysis of 10 different entity relationship modeling notations used in text books and Computer Assisted Systems Development (CASE) tools. The weak entity structure was represented in 5 of the notations and partially represented in another 3 notations.

Figure 9 shows an example of a weak entity. In order to uniquely identity a dependent of an employee we need the employee entity to exist in the system. A dependent of an employee cannot be determined without having an employee record i.e. without an employee the dependent records become invalid in the database, and have no significance of their own. The representation for a weak entity is to boxed rectangle i.e. rectangle in a rectangle.

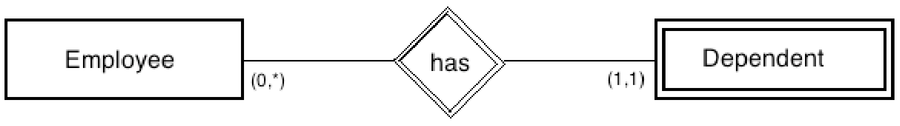

Figure 9 - Dependent Weak Entity – look here notation

There is no direct equivalent of a weak entity in a UML class diagram. However, there are several possible representations. In this example, a student may have many next-of-kin and a next-of-kin will be associated with only one student. Figure 10 shows one possibility for representing a weak entity on a UML diagram.

3.1.1 First possibility – Use Aggregation in UML

We could use aggregation for a weak entity concept. A possible depiction using composition aggregation could be as below in figure 10.

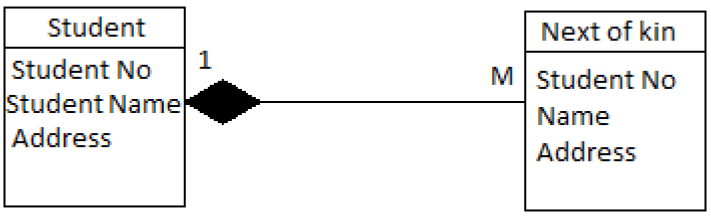

Figure 10 Possible notation for Weak Entity

We have used composition as if a student is deleted we do not need to keep the next-of-kin details. Although, one could probably argue here for having a regular aggregation (non-shaded diamond instead).

Peter Chen's original paper said that the weak entity can also "be applied recursively ". In our opinion this is equivalent to a common modelling pattern in where we have a chain of weak entities as in the example below and also as illustrated by the Flight, Flight-Leg, Flight-Leg Instance example in Elmasri and Navathe [2]. The example below in figure 11 shows how a University module is made up of components and a component can be made up of many elements. Students take the individual elements which are then put together to form component etc. Some modules may have only one component (maybe one exam), some have two components (maybe exam and coursework), some have components broken down into elements (maybe where there are several practical tasks which are assessed for a component and each task is regarded as an element.). This is a common pattern for University modules. Figure 11 shows the representation in UML and Figure 8 in EER using Elmasri and Navathe notation.

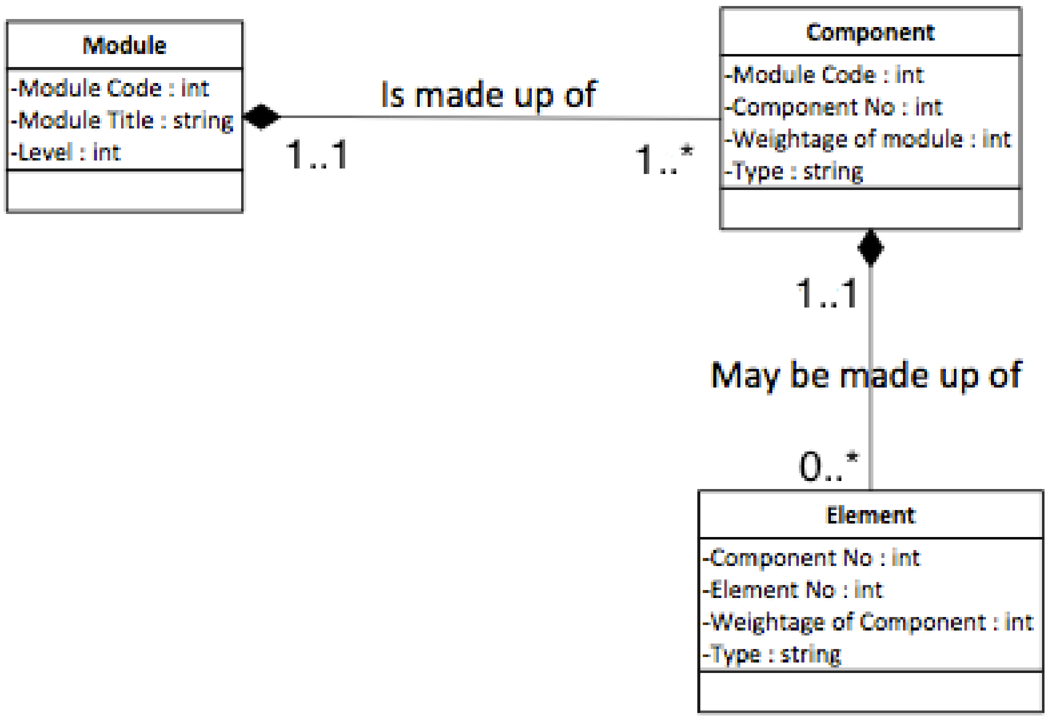

Figure 11 – Aggregation

The example in figure 8 works better using composition aggregation than the previous example with Student and next-of-kin as Modules are made up of components and then components made up of elements (whereas Students are not made up of next-of-kin!).

3.1.2 Use a UML Qualified Association – 2nd possibility

Qualifiers are used in cases where some piece of information can be used as a key to uniquely identify one out of a set of objects. The relevant properties of a key are that in a given context each key value can appear only once, and must somehow identify a single object, which is described by the key [20]. In UML a qualified association is "An association in which the objects in a "many" role are partially or fully disambiguated by an attribute called the qualifier" [21].

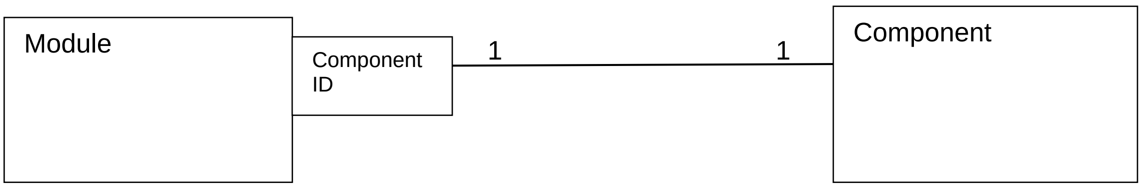

Figure 13 Weak Entity as a qualified association.

When using a qualified association the multiplicity is reduced to 1 at the component class, in our example, instead of what it previously was i.e. 1…*. With a qualified association the partial key is placed in a box attached to the Owner class/entity as in Figure 3 above. In our opinion this seems to over-ride the original 1:m relationship.

Elmasri and Navathe show a weak entity as a qualified association as in the above example [2] but also say it could be modelled as a qualified aggregation. Connolly and Begg describe a weak entity but do not go into any detail [3]. We leave it as an exercise to the reader to draw the module, component, element chain in an ER diagram using qualified associations. Blaha and Premerlani [21] provide a similar example which they refer to as a qualification cascade which is equal to a chain of weak entities.

3.1.3 3rd Possibility Ignore the weak entity and present it as an ordinary class

A further solution to depict a weak entity in a class diagram is to draw it using the simple class symbol and have the association between the two entities as if Next of Kin were a normal entity i.e. a strong entity. This way we can show both the association between the entities and the cardinalities between them.

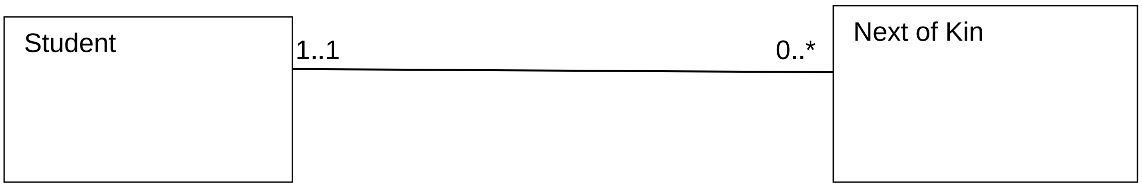

Figure 12 Weak Entity as a regular 1:m relationship.

3.1.4 4th Possibility - Show it as a function/method in a UML class diagram – 4th possibility

Another way to show the relationship between Student and Next of Kin is depicted in the diagram below. This approach in UML illustrates the relationship between the two entities in a better manner, and certainly eliminates the misconception which, using aggregation symbol for depicting weak entity would have raised.

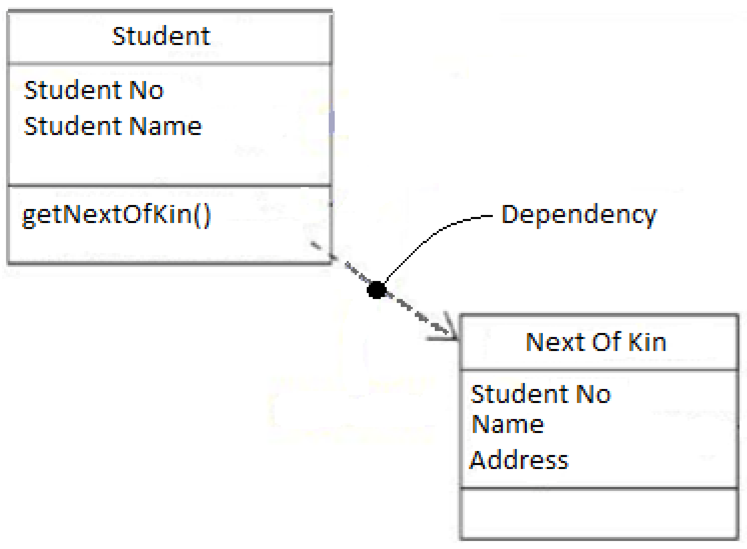

Figure 14

The dependency between the two entities is depicted by using one of the concepts that only belongs to Class Diagrams i.e. introduction of a behaviour getNextOfKin() which should get the information for a student's next of kin. From an implementation point of view in a database it could be a stored as a procedure or a tabular function using SQL. But still this representation lacks the cardinality between the two entities and neither does it is make it significant that Next of Kin's existence is dependent on the parent entity. Also it is introducing behavioural aspects into an otherwise static model thus moving from a conceptual level to an implementation specific (logical) model.

Therefore, in UML we have at least four different ways to show something that could be shown in one notation in EER. This makes it difficult for students and also makes it difficult to mark. Also the choice of which of the possible four ways to choose is not obvious.

3.2 Structural constraints

The second problem for novice database designers is the way that structural constraints are depicted on the diagrams. The various different ways and combinations are well documented in Song's paper [10] and some variations have been illustrated above in figures 4, 6 and 9. UML uses a "look across" direction for reading the constraints of cardinality and partiality with min and max numbers, whereas Elmasri and Navathe [2] use a "look here" notation with min and max numbers. It is a pity that the OMG group when designing UML did not choose the "look here" notation which was popular at the time and used in the Elmasri and Navathe [2] text book and also in Merise, the general purpose modelling methodology widely used in France. When marking student work we have noticed that sometimes students use "look here" and sometimes "look across" when adding the structural constraints to UML class diagrams. In many ways it does not matter as long as they are consistent (and they are not using a CASE tool which automatically maps to an implementation). Where a student shows a lack of understanding is if they mix both look across and look here on the same diagram.

Figure 16 and 17 below show two examples of mixed notation.

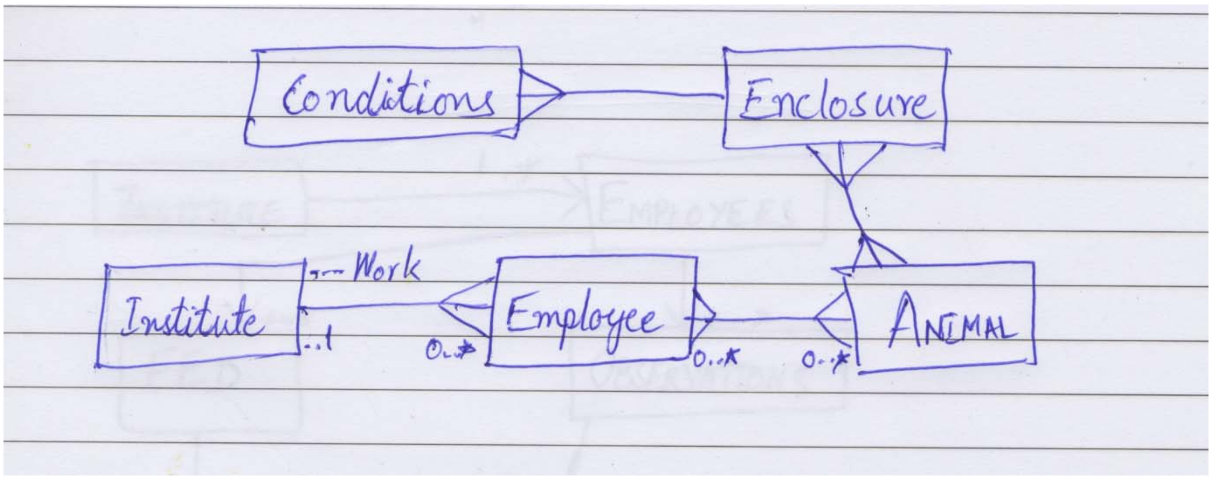

Figure 15

A 2nd year undergraduate student generated the solution in the image above, where he was asked to use UML class notation to draw an ER model for a provided case study. (Unfortunately this is often the terminology that is used “using a UML class diagram notation draw an ER diagram for...”) This particular group of students have been taught a crow’s feet notation in the first year followed by UML class diagram notation in the second year. The errors made by the student can be identified as follows:

The notation for a Class is used from the ER model instead of UML notation.

Crow’s feet have been used along with the UML notation for cardinality (0..*)

The majority of the relationship/association names have been left out.

The image below in figure 17 was also generated as part of an assignment which students had to do in the 2nd year of undergraduate study. Errors that can be identified in this solution are similar to the one provided earlier.

In figure 16 the student has used notation for an entity from the ER model instead of UML, whereas they were asked to make use of UML. The student has used arrows instead of crow’s feet. Using arrows as above are neither part of ER nor the UML model. UML model does have a symbol of arrows, but that is to identify the specialisation and generalisation in UML. In the case study above there were no super/sub classes.

Cardinalities have been used along with an attempt to use crow’s feet (instead of which arrows have been used). The student seems confused about the use of cardinality notation in UML. It seems as if (*..*) in the above diagram was used in a sense where student was trying to show many-to-many relationship. It looks like as if the first asterisk in (*..*) between “Enclosures” and “Conditions” shows that there is a many to many relationship between the two entities. Similarly looking at the entities “Animals” and “Enclosures” we see that the cardinalities are shown as *..1 may be trying to show one-to-many relationship with “Animals” have the many end of the relationship.

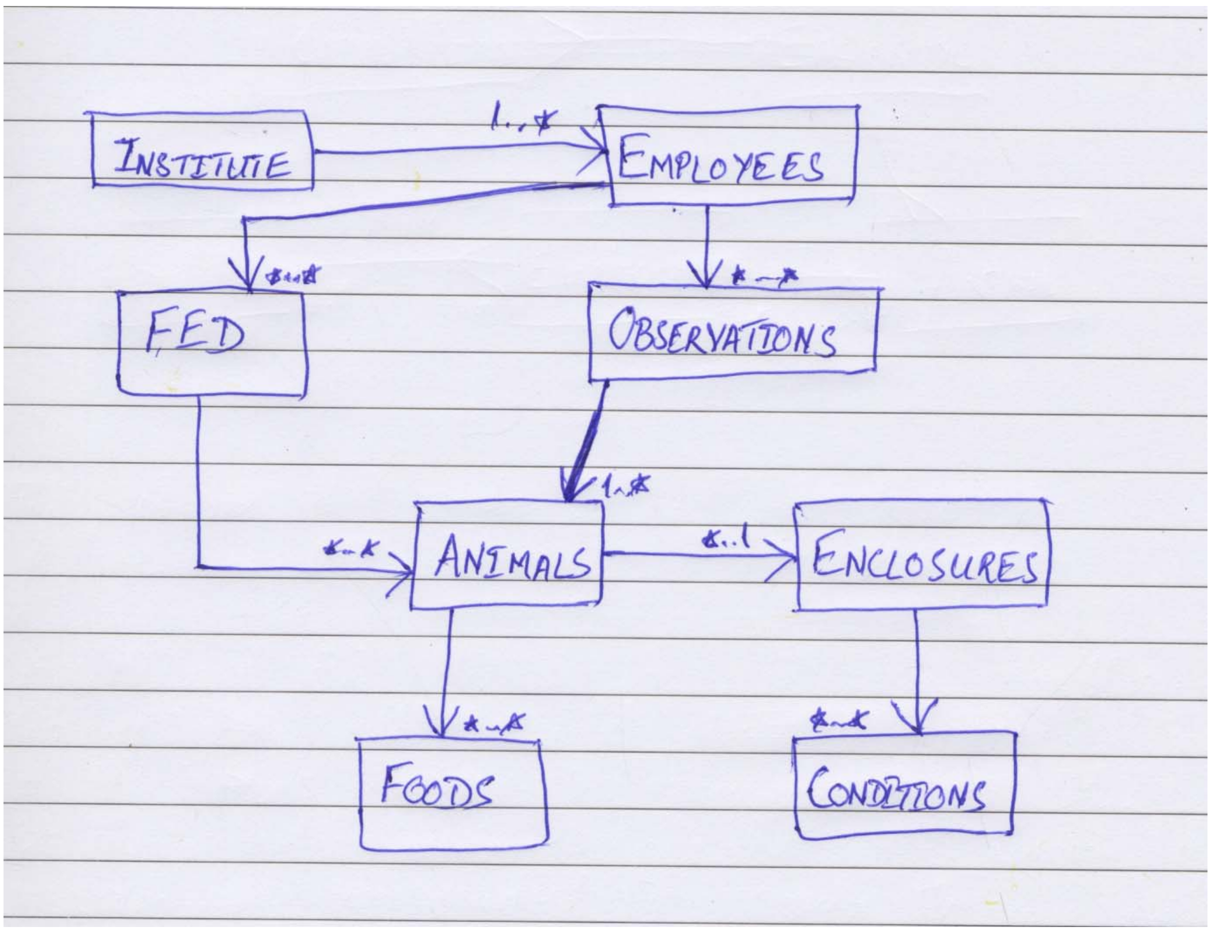

Figure 16

The solutions provided in figure 15 and 16 show that there is some confusion created by the use of ER and UML notations at 1st and 2nd year respectively for undergraduate students.

4. CURRENT TEXTBOOKS.

Apart from the fact that there are various different variations on notations for the ER model there is also a confusion of terminology. Connolly & Begg [3] in their chapter on Entity Relationship modelling state "*Although we use the UML notation for drawing ER models, we continue to describe the concepts of ER models using traditional database terminology*" (page 322).

Several of the popular text books at the moment use ER or UML or both for teaching database modelling.

	Connolly & Begg 5th edition.	**Elmasri & Navathe 3rd edition**	**Elmasri & Navathe 6th edition**	**C J Date 8th edition**	**Ramakrishnan & Gehrke**
ER	Appendix	Yes	Yes	Yes	Yes
UML	Yes (but described in ER terms)	Appendix	Yes	No	Short description

Therefore if a student picks up a text book they could be faced with either; one of the many ER and EER variations, UML, UML and a variation of ER covered separately, or UML and ER merged together. The authors would recommend the point of view taken by Blaha [16] who writes:

"*The UML data structure model is just another Chen dialect, but one that has the backing of a standard…… The UML is normally used in conjunction with object-oriented jargon which I avoid. Object-oriented jargon connotes programming which I do not intend. This book's focus is on data modelling.*"

We find this approach refreshing. We can throw out all the OO side of it and use UML as an Abstract model. Blaha continues to say that he uses UML for the conceptual stage and then moves on to IDEFIX notation for a logical stage with more detail. "*The UML is good for abstract modelling and not for database design. IDEFIX is good for database design and not for abstract modelling. Both notations are useful, but each has its place.*" However we have highlighted two aspects in this paper which can cause problems for students when trying to produce a conceptual model with UML. Firstly how to represent a weak entity type structure in UML and secondly where to put the constraints. Also a UML class diagram is not as clear as an ER diagram at the Requirements Analysis stage when discussion takes place with end users.

5. CONCLUSION

In order to conclude whether UML class diagrams can be used for the purposes of database design and if there is any possibility to substitute them in place of ERDs, we need to think of the issues identified earlier on in this chapter. We also need to consider the levels of abstraction that both modelling techniques provide along with the way these techniques are used and the professionals who use them.

UML Class Diagrams not only depict entities with their attributes, they also have the power to depict the domains these attributes belong to along with the relationships and other participation constraints, which is quite helpful when designing physical level schemas. Class Diagrams can also be used to show the behaviours, in case of databases, this feature can be used to show low level details about the stored procedures and functions that would have access to the entities or perform operations like insert, update and delete etc. All of this means a great level of detail which is normally necessary at physical level of database design. Therefore we could argue that the UML class diagram is more of a logical model (that is implementation specific). ERDs on the other hand, are at a conceptual level of database design and at a higher level of abstraction dealing with the main items and their relationships and not with implementation specific detail.

We need to bear in mind that primary purpose for UML Class diagrams is to depict Classes and their associations in Object Oriented paradigm. The concepts of OO classes differ from that of entities, in ERD, in that classes may have member properties and may also have behaviours. They also have the concepts of making their members private and public. UML Class diagrams are more of a programmer's tool than a database administrator's tool. Programs and database have different structures and have differences in how things work internally in programs and databases, especially relational databases

As regards a weak entity it is difficult to choose an appropriate representation on a UML class diagram. Weak entities can differ; Byrne and Garvey [11] illustrate different types. Generally speaking weak entities can either just link to one parent entity (as in the Student/Next-of-Kin example) or be in a chain (as in the Module, Component, and Element example). Sometimes a qualified association is appropriate to represent a single weak entity with a parent entity and sometimes aggregation could be used. (However we would argue that the UML qualified association concept is overly complicated anyway.) Aggregation can represent a chain of weak entities where the type of relationship is a "part making up a whole" type of relationship.

Geolman, D. and Song, Y. [12], state that we need to make sure that we are not empowering ERD or any other Modelling technique with too many additional features which will make it difficult for users to understand and will be an unnecessary burden on those models. They will lose their visual power in communicating the structure of the database in general. Complex constraints should either be written down in text or be elaborated via separate low level models, focusing on individual parts of the huge complex models.

We would argue that the UML class diagram is over-loaded with features and not suitable for a conceptual model.

We suggest a good method of teaching would be crows feet notation with a binary model in the first year, EER in the second year, UML class diagrams as level 6 or when moving from conceptual (not implementation specific) to a logical design (implementation specific). In future studies we intend to look at many-to-many, ternary and aggregation representations in UML and EER.

Finally we would like to finish with an example of the work of a level 5 student. When asked to draw a global ER diagram for an in-class test (using a UML class diagram) she drew a draft diagram using crows feet notation and then a final version in a UML class diagram. Therefore she drew a conceptual model first (ER) and then a more detailed logical model (UML class diagram.) This is how we should teach. Crows feet for initial Conceptual binary model – EER for the added concepts of specialisation and aggregation and ternary relationships, and UML for a logical/physical model.

6. REFERENCES

[1] Chen, P. (1976) '*The Entity-Relationship Model – Towards a Unified View of Data'*. ACM Transactions on Database Systems [Online]. March. pp. 9-36. Available at: http://csc.lsu.edu/news/erd.pdf [Accessed: 17 February, 2013].

[2] Elmasri, R. Navathe, S. B. (2011) *Database Systems Models, Languages, Design and Application Programming.* Sixth Edition*.*

[3] Connolly, T. and Begg, C. (2010) *Database systems*. Boston, Mass. [u.a.]: Addison-Wesley.

[4] Batra D, Davis J "Conceptual data modeling in database design". Int. J. Man-Machine Studies (1992) 37, 83-101 Academic Press Limited.

[5] Moody D, Shanks G. "What makes a Good Data Model? Evaluating the Quality of Entity Relationship Models." Proceedings 13th International Conference on ER Approach 1994 Manchester Springer-Verlag.

[6] Kesh S Evaluating the quality of entity relationship models. Information and Software Technology 1995 37 (12) 681-689.

[7] Saiedian, H. "An evaluation of extended entity-relationship model". Information and Software Technology 39 (1997) 449-462

[8] Jones T, Song I Y "Analysis of binary/ternary cardinality combinations in ER modeling". Data and Knowledge Engineering 19 (1996) 39-64

[9] Siau K, Wand Y, Benbasat I. "The Relative Importance of Structural Constraints and Surface Semantics in Information Modeling". Information Systems Vol. 22 No. 2/3 pp 155-170 1997 Elsevier Science Ltd.

[10] Song I, Evans M, Park E, A Comparative Analysis of Entity-relationship Diagrams. Journal of Computer & software Engineering, 3(4), 427-459 (1995)

[11] Byrne B, Garvey M *Weak Entities in Conceptual Modeling* UKAIS (2006), *Gloucestershire*

[12] Song Il-Y, Goelman D *Entity-Relationship Modeling Re-revisited (2004) Springer Verlag, Lecture notes and Computer Science, 3288, pg 43-52.*

[13] Urban S, Dietrich S, *Using UML Class Diagrams for a Comparative Analysis of Relational, Object-Oriented and Object Relational Database Mappings.* Proceedings of SIGCSE'03 February, pp 19-23 2003 Reno, Nevada.

[14] Martin, F. & Kendall, S (1999) *UML Distilled: A Brief Guide to the Standard Object Modelling Language*, Second Edition. Addison-Wesley.

[15] Siau, K. and Halpin, T. (2001) *Unified modeling language*. Hershey, Pa.: Idea Group Pub.

[16] Blaha M. (2010) Patterns Of Data Modelling. CRC Press, Pg 2.

[17] Booch, G. and Rumbaugh, J., et al. (1999) *The unified modeling language user guide*. Reading Mass.: Addison-Wesley.

[18] Balaban M, Shoval P Resolving the "Weak Status" of Weak Entity Types in Entity Relationship Schemas. Proceedings of the Entity Relationship Conference 1999 Paris, France p369-383 Springer Lecture Notes in Computer Science

[19] Thalheim B Entity Relationship Modeling. Foundations of Database Technology. Springer –Verlag New York 2000. p36, p40

[20] Mark Priestley (2003), Practicle Object Oriented Design with UML. Mc Grawhill, Second Ed.

[21] Blaha M, Premerlani W *Object-Oriented Modeling and Design for Database Applications* Prentice Hall 1998