UML Data Models From An ORM (Object-Role Modeling) Perspective. Data Modeling at Conceptual Level

Lecturer Ph. D. Candidate DANIEL IOAN HUNYADI,
Lecturer Ph. D. Candidate MIRCEA ADRIAN MUSAN
Department of Computer Science and Economic Informatics
University "Lucian Blaga" of Sibiu
Sibiu, Bd. Calea Dumbravii 3-5
ROMANIA
daniel.hunyadi@ulbsibiu.ro, mircea.musan@ulbsibiu.ro

Abstract: - This paper provides an overview of Object-Role Modeling (ORM), a fact-oriented method for performing information analysis at the conceptual level. It provides both a graphical and textual languages and a procedure which guides the use of the languages.
The article is structured in two main parts. The first part presents an overview of ORM along a real example, while the second part of the article makes a comparison between ORM and UML from the conceptual data modeling perspective.
This paper examining data modeling in the Unified Modeling Language (UML) from the perspective of Object Role Modeling (ORM). It provided some historical background on both approaches, identified design criteria for modeling languages, and discussed how object reference and single-valued attributes are modeled in both. It compared UML multi-valued attributes with ORM relationship types, and discussed basic constraints on both, as well as instantiation using UML object diagrams or ORM fact tables. This third issue compares UML associations and related multiplicity constraints with ORM relationship types and related uniqueness, mandatory role and frequency constraints. It also contrasts instantiation of associations using UML object diagrams and ORM fact tables.

Key-Words: - Object-Role Modeling (ORM), FORML (Formal Object-Role Modeling Language), ER diagrams, CSDP, abstraction mechanism, semantic stability, semantic relevance, formal foundation.

1 Introduction

It is well recognized that the quality of a database application depends critically on its design. To help ensure correctness, clarity, adaptability and productivity, information systems are best specified first at the conceptual level, using concepts and language that people can readily understand. The conceptual design may include data, process and behavioral perspectives, and the actual DBMS used to implement the design might be based on one of many logical data models (relational, hierarchic, network, object-oriented etc.).
This overview focuses on the data perspective, and assumes the design is to be implemented in a relational database system.
Designing a database involves building a formal model of the application area or universe of discourse (UoD). To do this properly requires a good understanding of the UoD and a means of specifying this understanding in a clear, unambiguous way.

Object Role Modeling (ORM) is a powerful method for designing and querying database models at the conceptual level, where the application is described in terms easily understood by non-technical users. In practice, ORM data models often capture more business rules, and are easier to validate and evolve than data models in other approaches.
Object-Role Modeling (ORM) simplifies the design process by using natural language, as well as intuitive diagrams which can be populated with examples, and by examining the information in terms of simple or elementary facts. By expressing the model in terms of natural concepts, like objects and roles, it provides a conceptual approach to modeling.
Early versions of object-role modeling were developed in Europe in the mid-1970s (e.g. binary relationship modeling and NIAM). The version discussed here is based on the author's formalization of the method, and incorporates extensions and refinements arising from research conducted in Australia and the USA. The associated language

FORML (Formal Object-Role Modeling Language) is supported in Microsoft Visio for Enterprise Architects (VEA), part of Visual Studio .NET Enterprise Architect.
Another conceptual approach is provided by Entity-Relationship (ER) modeling. Although ER models can be of use once the design process is finished, they are less suitable for formulating, transforming or evolving a design. ER diagrams are further removed from natural language, cannot be populated with fact instances, require complex design choices about attributes, lack the expressibility and simplicity of a role-based notation for constraints, hide information about the semantic domains which glue the model together, and lack adequate support for formal transformations. Many different ER notations exist that differ in the concepts they can express and the symbols used to express these concepts. For such reasons we prefer ORM for conceptual modeling. In addition to ORM, VEA supports IDEF1X (a hybrid of ER and relational modeling) as a view of ORM.
Although the detailed picture provided by ORM diagrams is often desirable, for summary purposes it is useful to hide or compress the display of much of this detail. Though not discussed here, various abstraction mechanisms exist for doing this. If desired, ER diagrams can also be used for providing compact summaries, and are best developed as views of ORM diagrams. This overview conveys the main ideas in ORM by discussing a case study. First we explain the stepsused to develop a conceptual design. To help communicate the ideas, we deliberately make some mistakes, and later show how the design method helps to correct these errors. We also include a simple example to show how the conceptual design may be “optimized” for relational systems by applying a transformation.
An algorithm for mapping this design to a normalized, relational database schema is then outlined.
With VEA, the conceptual design can be entered in either graphical or textual form, and automatically mapped to a relational schema for use in a variety of relational DBMSs. Finally, a brief sketch is given of how ORM may be used as a sound basis for conceptual queries. For a detailed discussion of ORM, see [1].

2 The Conceptual Schema Design Procedure

The information systems life cycle typically involves several stages: feasibility study; requirements analysis; conceptual design of data and operations; logical design; external design; prototyping; internal design and implementation; testing and validation; and maintenance. ORM's conceptual schema design procedure (CSDP) focuses on the analysis and design of data. The conceptual schema specifies the information structure of the application: the types of fact that are of interest; constraints on these; and perhaps derivation rules for deriving some facts from others.
With large-scale applications, the UoD is divided into convenient modules, the CSDP is applied to each, and the resulting subschemas are integrated into the global conceptual schema. The CSDP itself has seven steps. The rest of this section illustrates the basic working of this design procedure by means of a simple example.
The conceptual schema design procedure (CSDP) in step and description:
1. Transform familiar information examples into elementary facts, and apply quality checks
2. Draw the fact types, and apply a population check
3. Check for entity types that should be combined, and note any arithmetic derivations
4. Add uniqueness constraints, and check arity of fact types
5. Add mandatory role constraints, and check for logical derivations
6. Add value, set comparison and subtyping constraints
7. Add other constraints and perform final checks

Step 1 is the most important stage of the CSDP. Examples of the kinds of information required from the system are verbalized in natural language. Such examples are often available in the form of output reports or input forms, perhaps from a current manual version of the required system. If not, the modeler can work with the client to produce examples of output reports expected from the system. To avoid misinterpretation, it is usually necessary to have a UoD expert (a person familiar with the application) perform or at least check the verbalization. As an aid to this process, the speaker imagines he/she has to convey the information contained in the examples to a friend over the telephone. For our case study, we consider a fragment of an information system used by a university to maintain details about its academic staff and academic departments. One function of the system is to print an academic staff directory, as exemplified by the report extract shown in Table 2. Part of the modeling task is to clarify the meaning of terms used in such reports. The descriptive narrative provided here would thus normally be derived from a discussion with the UoD expert. The terms

"empNr" and "extNr" abbrevi-ate "employee number" and "extension number".

Step 2 of the CSDP is to draw a draft diagram of the fact types and apply a population check (see Figure 1). Entity types are depicted as named ellipses. Predicates are shown as named sequences of one or more role boxes. Predicate names are read left-to-right and top-to-bottom, unless prepended by "<<" (which reverses the reading direction). An n-ary predicate has n role boxes. The inverse predicate names have been omitted in this figure. Value types are displayed as named, broken ellipses. Lines connect object types to the roles they play. Reference modes are written in parenthesis: this is an abbreviation for the explicit portrayal of reference types. For example, the notation "Academic (empNr)" indicates an injection (1:1-into mapping) from the entity type Academic to the value type EmpNr. In this example there are seven fact types. As a check, each has been populated with at least one fact, shown as a row of entries in the associated fact table, using the data from rows 1 and 3 of Table 2. The English sentences listed before as facts f1-f7, as well as other facts from row 3, may be read directly off this figure. Though useful for validating the model with the client and for understanding constraints, the sample population is not part of the conceptual schema diagram itself.

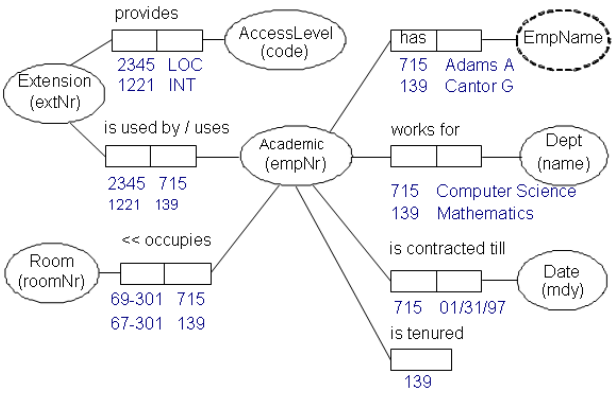


Fig. 1 Draft diagram of fact types for Table 2 with sample population

This leads us to **Step 3** of the CSDP: check for entity types that should be combined, and note any arithmetic derivations. The first part of this step prompts us to look carefully at the fact types for f11, f15 and f17. Currently these are handled as three ternary fact types: *Professor obtained Degree from University;SeniorLecturer obtained Degree from University; Lecturer obtained Degree from University*. The common predicate suggests that the entity types Professor, SeniorLecturer and Lecturer should be collapsed to the single entity type Academic, with this predicate now shown only once, as shown in Figure 2.

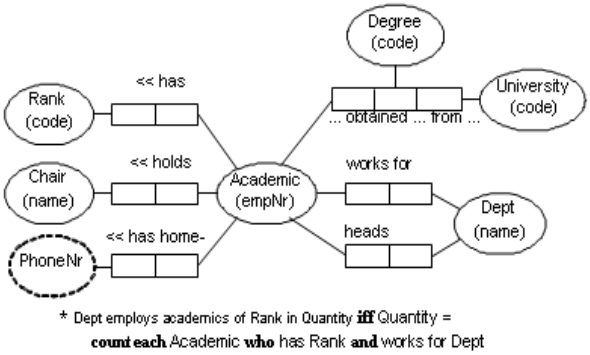


Fig.2 Extra fact types needed to capture the additional information

The second aspect of Step 3 is to see if some fact types can be derived from others by arithmetic. Since we now record the rank of academics as well as their departments, we can compute the number in each rank in each department simply by counting. So facts like f9' are derivable. If desired, derived fact types may be included on a schema diagram if they are marked with an asterisk "*" to indicate their derivability. To simplify the picture, it is usually better to omit derived predicates from the diagram. However in all cases a derivation rule must be supplied. This may be written below the diagram (see Figure 2). Here "iff" abbreviates "if and only if".

Step 4 of the CSDP is to add uniqueness constraints and check the arity of the fact types. Uniqueness constraints are used to assert that entries in one or more roles occur there at most once. A bar across n roles of a fact type (n > 0) indicates that each corresponding n-tuple in the associated fact table is unique (no duplicates are allowed for that column combination). Arrow tips at the ends of the bar are needed if the roles are non-contiguous (otherwise arrow tips are optional). A uniqueness constraint spanning roles of
different predicates is indicated by a circled "u": this specifies that in the natural join of the predicates, the combination of connected roles is unique. For example, a fragment of the conceptual schema under consideration is displayed in Figure 3. While these constraints are suggested by the original population, the domain expert should normally be consulted to verify them. It is sometimes helpful to construct a test population for each fact type in this regard, though simple questions are usually more efficient. The internal uniqueness constraints on the binary fact types assert that each academic has at most one

rank, holds at most one chair (and vice versa), works for at most one department, and has at most one employee name.

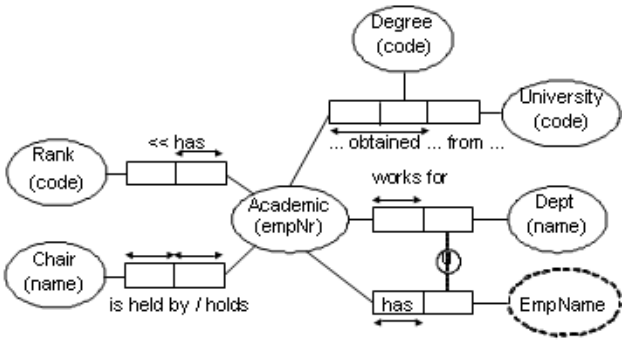


Fig. 3 Some of the fact types, with uniqueness constraints added

If a fact type is elementary all its functional dependencies (FDs) are implied by uniqueness constraints. For example, each academic has only one rank (hence in the fact table for Academic has Rank, entries in the rank column are a function of entries in the academic column). If in doubt, one checks for FDs not so implied; if such an FD is found, the fact type is split on the source of the FD.

Step 5 of the CSDP is to add mandatory role constraints, and check for logical derivations. A role is mandatory (or total) for an object type if and only if every object of that type which is referenced in the database must be known to play that role. This is explicitly shown by means of a mandatory role dot where the role connects with its object type. If two or more roles are connected to a circled mandatory role dot, this means the disjunction of the roles is mandatory (i.e. each object in the population of the object type must play at least one of these roles) – an inclusive-or constraint.

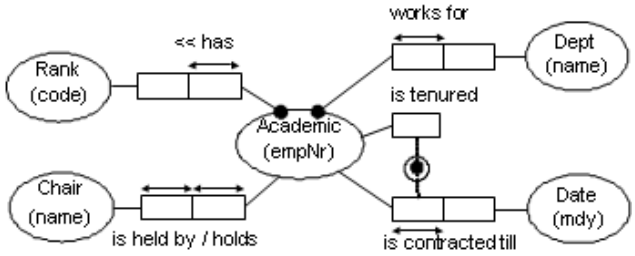


Fig.4 Some of the fact types, with mandatory role constraints added

For example, Figure 4 adds mandatory role constraints to some of the fact types already discussed.
These dots indicate that each academic has a rank and works for a department; moreover each academic either is tenured or is contracted till some date. Roles that are not mandatory are optional. The role of having a chair is optional. The roles of being contracted or being tenured are optional too, but their disjunction is mandatory. If an object type plays only one fact role in the global schema, then by default this is mandatory, but a dot is not shown (e.g. the role played by Rank is mandatory by implication).
Now that uniqueness and mandatory role constraints have been discussed, reference schemes can be better understood. Simple reference schemes involve a mandatory 1:1 mapping from entity type to value type. For example, the notation "Rank (code)" abbreviates the binary reference type: Rank has Rankcode. If shown explicitly, both roles of this binary have a simple uniqueness constraint, and the reference role played by Rank has a mandatory role dot. With composite reference, a combination of two or more values can be used to refer to an entity. For example, while EmpNr provides a simple primary identifier for Academic, the combination of Dept and EmpName provides a secondary identification scheme. Sometimes composite schemes are used for primary reference. The second stage of Step 5 is to check for logical derivations (i.e. can some fact type be derived from others without the use of arithmetic?). One strategy here is to ask whether there are any relationships (especially functional relationships) which are of interest but which have been omitted so far. Another strategy is to look for transitive patterns of functional dependencies. For example, if an academic has only one phone extension and an extension is in only one room, we could use these to determine the room of the academic. However, for our application the same extension may be used in many rooms, so we discard this idea.

In **Step 6** of the CSDP we add any value, set comparison and subtyping constraints. Value constraints specify a list of possible values for a value type. These usually take the form of an enumeration or range, and are displayed in braces besides the value type or its associated entity type. For example, Rankcode is restricted to {'P','SL','L'} and AccessLevelcode to {'INT','NAT','LOC'}. These are displayed in the global conceptual schema (Figure 5). Set comparison constraints specify subset, equality or exclusion constraints between compatible roles or sequences of compatible roles. A subset constraint from one role sequence to another indicates that the population of the first must always be a subset of the second, and is denoted by a circled with a dotted arrow from source to target In Figure 5, a pair-

subset constraint runs from the heads predicate to the works for predicate, indicating that a person who heads a department must work for the same department.

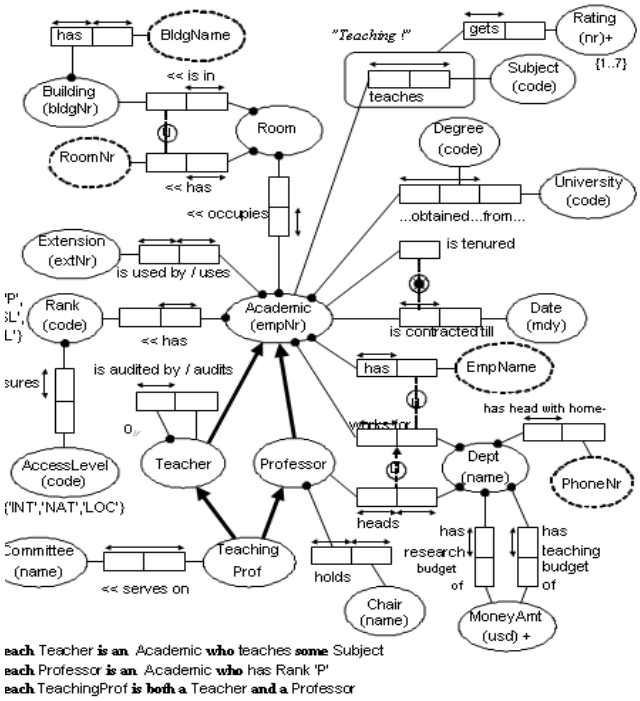


Fig. 5 The final conceptual schema

Step 7 of the CSDP adds other constraints and performs final checks. We briefly illustrate two other constraints. The audits fact type has both its roles played by the same object type (this is called a ring fact type). Theoir notation beside it indicates the predicate is irreflexive (no teacher audits himself/herself).Suppose we also need to record the teaching and research budgets of the departments. We might schematize this as in Figure 6. Here the "2" beside the role played by Dept is a frequency constraint indicating that each department that is included in the population of that role must appear there twice. In conjunction with the other constraints, this ensures that each department has both its teaching and research budgets recorded.

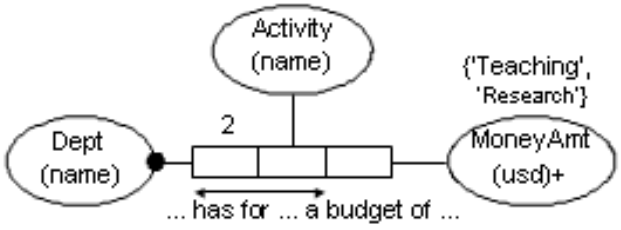


Fig. 6 Each department has two budgets

The CSDP ends with some final checks that the schema is consistent with the original examples, avoids redundancy, and is complete. No changes are needed for our example. There is a minor derived redundancy, since if someone heads a department, we know from the subset constraint that this person works for that department; but this is innocuous. Other schematizations are possible (e.g. we can define works in and heads to be pair-exclusive, or use a unary is head instead of the binary heads) but we ignore these alternatives here.

Once the global schema is drafted, and the target DBMS decided, various optimizations can usually be performed to improve the efficiency of the logical schema that results from the mapping. Assuming the conceptual schema is to be mapped to a relational database schema, the fact type in Figure 6 will map to a separate table all by itself (because of its composite uniqueness constraint). Since some other information about departments is mapped to another table, if we want to retrieve all the details about departments in a single query we will have to perform a table join. Joins tends to slow things down.

Moreover, we probably want to compute the total budget of a department, and with the current schema this will involve a self-join of the table since the details of the two budgets are on separate rows. We can avoid all these problems by transforming the ternary fact type in Figure 6 into the following two binaries before we map: Dept has teaching budget of MoneyAmt; Dept has research budget of MoneyAmt. These binaries have simple keys, and will map to the "main" department table. Another optimization may be performed which moves the home phone information to Dept instead of Professor, but the steps underlying this are a little advanced, so we ignore a detailed discussion of this move here. Figure 6 includes both these revisions to the conceptual schema.

3 Study Case

To give you a real example of applying ORM in data modeling we will shortly present a fragment of a project in which we were involved.

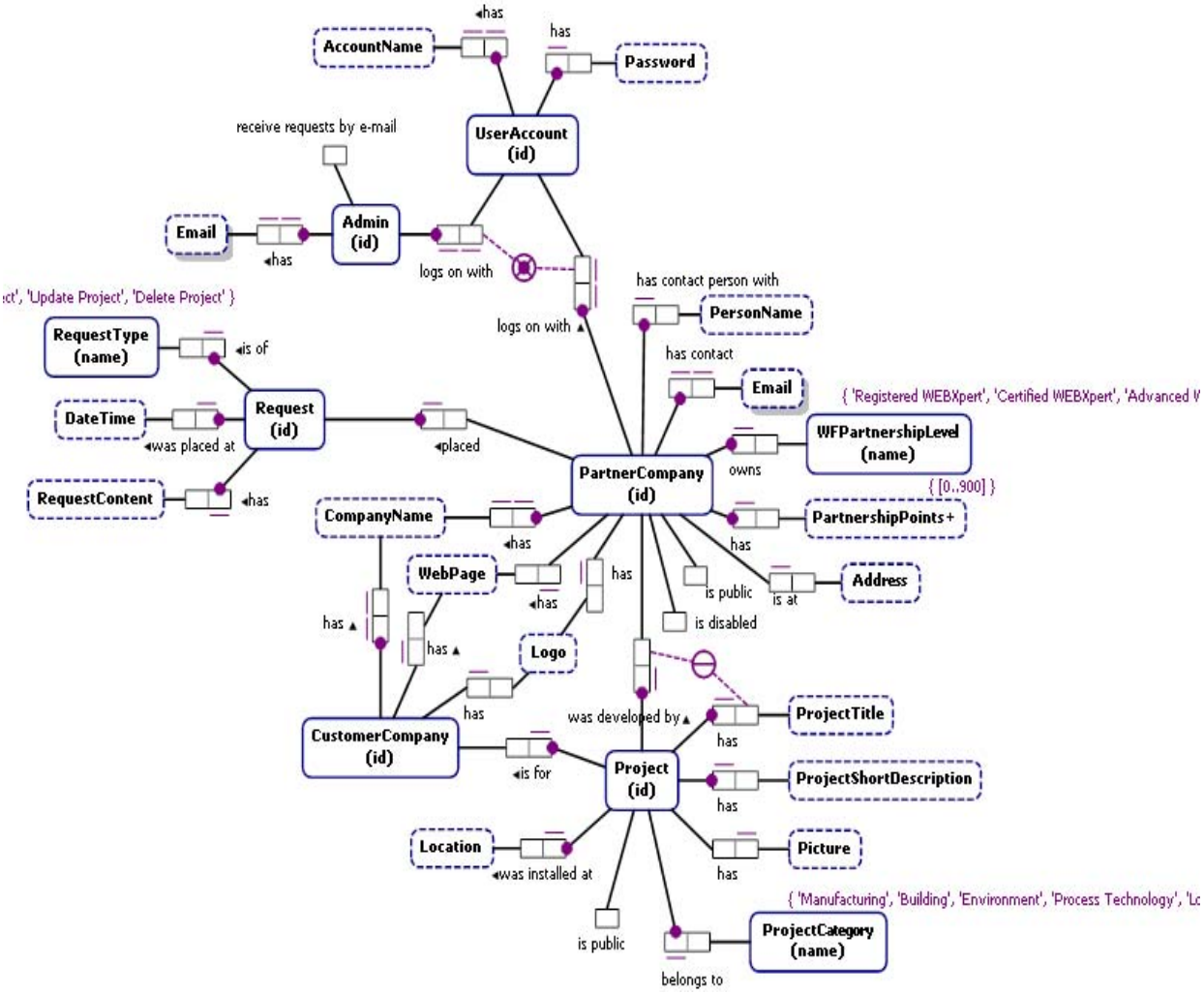
It's about a web portal by which a company presents online the projects it developed over the time for different customers. We used the ORM methodology to create the structure of the relational database used by the portal to store projects information.

Applying the steps of the conceptual schema design procedure the following ORM diagram is presented in next page.

Next, we will shortly present the steps of CSDP. As the first step, the knowledge about the application

domain (the universe of discourse) was transformed in elementary facts that further were represented graphically on the diagram.
Here are some examples of elementary facts:
1. Project has Title.
2. Company has Address.
3. Project was developed by Company.
Next step was to apply the data constraints over the elementary facts and represent them on the diagram.
1. Each Project has exactly one Title.
2. Each Company has at most one Address.
3. Is it possible that a Company has no Address.
4. Each Project was developed by exactly one Company.
After the ORM diagram was finalized, we applied the algorithm of mapping the diagram to the relational database structures. The algorithm ensures that the resulting database structure conforms to the conditions of the 5th database normalization form (5NF).
Once the conceptual schema has been specified, a simple algorithm is used to group these fact types into normalized tables. If the conceptual fact types are elementary (as they should be), then the mapping is guaranteed to be free of redundancy, since each fact type is grouped into only one table, and fact types which map to the same table all have uniqueness constraints based on the same attribute(s).

Applying the steps of the conceptual schema design procedure the following ORM diagram resulted:



In this study of case we present an ORM diagram witch results from this project.

The diagram models in graphical mode the domain of application representing all the types of information which knows about that application and

the coercion which applies those information. This application represents an on-line typecast with the projects which are developed by a firm. The application contains information about the projects developed by the firms and the customers of the respective projects. The purpose of this application is that to admit of any navigator of Internet to access the on-line portofolio of developed projects of the company. Also, the application admits of its user to create his account and creates his own own portofolio of projects developed with the company which has the site.

After the finalization of the ORM's diagram it administer of this algorithm of mapping of the diagram at the relation structure of database.

Like conclusion we can say that after we used another methods of modeling of data at the conceptual level like UML or ER, consider the the diagrams ORM are more expressive.

4 UML Data Models From An ORM Perspective

Although the Unified Modeling Language (UML) facilitates software modeling, its object-oriented approach is arguably less than ideal for developing and validating conceptual data models with domain experts. Object Role Modeling (ORM) is a fact-oriented approach specifically designed to facilitate conceptual analysis and to minimize the impact on change. Since ORM models can be used to derive UML class diagrams, ORM offers benefits even to UML data modelers. This multi-part article provides a comparative overview of both approaches.

In our competitive and dynamic world, businesses require quality software systems that meet current needs and are easily adapted. These requirements are best met by modeling business rules at a very high level, where they can be easily validated with clients, and then automatically transformed to the implementation level. The Unified Modeling Language (UML) is becoming widely used for both database and software modeling, and version 1.1 was adopted in November 1997 by the Object Management Group (OMG) as a standard language for object-oriented analysis and design [8, 9]. Initially based on a combination of the Booch, OMT (Object Modeling Technique) and OOSE (Object-Oriented Software Engineering) methods, UML was refined and extended by a consortium of several companies, and is undergoing minor revisions by the OMG Revision Task Force [7]. A simple introduction to UML is contained in [6], and a thorough discussion of OMT for database applications is given in [5], although its notation for multiplicity constraints differs from the UML standard.

UML includes diagrams for use cases, static structures (class and object diagrams), behavior (state-chart, activity, sequence and collaboration diagrams) and implementation (component and deployment diagrams). For data modeling purposes UML uses class diagrams, to which constraints in a textual language may be added. Although class diagrams may include implementation detail (e.g. navigation and visibility indicators), it is possible to use them for analysis by omitting such detail. When used in this way, class diagrams essentially provide an extended Entity Relationship (ER) notation. UML's object-oriented approach facilitates the transition to object-oriented code, but can make it awkward to capture and validate business rules with domain experts. This problem can be remedied by using a fact-oriented approach where communication takes place in simple sentences, and each sentence type can easily be populated with multiple instances. Object Role Modeling (ORM) is a fact-oriented approach that harmonizes well with UML, since both approaches provide direct support for roles, n-ary associations and objectified associations. ORM pictures the world simply in terms of objects (entities or values) that play roles (parts in relationships).

ORM originated in the mid-1970s as a semantic modeling method, one of the early versions being NIAM (Natural language Information Analysis Method), and has since been extensively revised by many researchers. Overviews of ORM may be found in [6, 7] and a detailed treatment in [5]. Although all versions of ORM are based on the same framework, minor variations do exist. This article focuses on the most popular version of ORM as supported in modeling and query tools such as Visio's InfoModeler and ActiveQuery. Since business requirements are subject to ongoing change, it is critical that the underlying data model be crafted in a way that minimizes the impact of these changes. The ORM framework is more stable under business changes than either OO or ER models, and facilitates the remaining changes that need to be made. This stability applies not only to the model itself, but also to conceptual queries based on the model.

Although ORM can be used independently of other methods, it may also be used in conjunction with them. To better exploit the benefits of UML, or ER for that matter, ORM can be used for the conceptual analysis of business rules, and the resulting ORM model can be easily transformed into a UML class diagram or ER diagram.

This article summarizes the main data modeling constructs in both ORM and UML, and discusses how they relate to one another. It aims to provide a basic understanding of both approaches and to illustrate translation between their notations. Along the way, some comparative advantages of ORM are noted.

However this is not to disparage UML, which does have some nice features. Overall, UML provides a useful suite of notations for behavior and software modeling, and its class diagram notation is better than most other ER notations for data modeling. Visio Professional already provides basic support for several data and process modeling notations, and the integration of InfoModeler technology will enable very powerful support for both ORM and UML. So it will be possible to work in one or more of your preferred notations (ORM, UML, ER) with automatic mapping to an implementation in a variety of DBMSs. You could even do part of the model in ORM and part in UML, and have these merged to a single model.

Some information modeling approaches allow instances of relationships or associations to be treated as entities in their own right. In the Unified Modeling Language (UML), this modeling technique is called “reification”, and is mediated by means of association classes. In Object-Role Modeling (ORM), this process is called “objectification” or “nesting”. While this modeling option is rarely supported by industrial versions of Entity-Relationship Modeling (ER), it is allowed in several academic versions of ER. Objectification is related to the linguistic activity of nominalization, of which two flavors may be distinguished: circumstantial; and propositional. In practice, objectification needs to be used judiciously, as its misuse can lead to implementation anomalies, and those modeling approaches that permit objectification often provide incomplete or flawed support for it.

To provide an evaluation framework, some design criteria for modeling languages are first identified. We then discuss simple cases of how objects are referenced, and how single-valued “attributes” and can be captured in ORM and UML. From an ORM perspective, we confine our discussion of constraints to simple uniqueness and mandatory role constraints. From a UML perspective, we consider only attribute multiplicity and related textual constraints.

Later parts will discuss UML associations and more advanced features such as other constraint types, aggregation, subtyping, derivation rules and queries.

4.1 Conceptual modeling language criteria

A modeling method comprises a language and also a procedure for using the language to construct models. Written languages may be graphical (diagrams) and/or textual. Conceptual models portray applications at a fundamental level, using terms and concepts familiar to the application users. In contrast, logical and physical models specify underlying database structures to be used for implementation, and external models specify user interaction details (e.g. design of screen forms and reports). The following criteria provide a useful basis for evaluating conceptual modeling methods:

- Expressibility
- Clarity
- Semantic stability
- Semantic relevance
- Validation mechanisms
- Abstraction mechanisms
- Formal foundation

The expressibility of a language is a measure of what it can be used to say. Ideally, a conceptual language should be able to model all conceptually relevant details about the application domain. This is called the 100% Principle [9]. Object Role Modeling is primarily a method for modeling and querying an information system at the conceptual level, and for mapping between conceptual and logical levels. Although various ORM extensions have been proposed for object-orientation and dynamic modeling, the focus of ORM is on data modeling, since the data perspective is more stable and it provides a formal foundation on which operations can be defined. In this sense, UML is generally more expressive than standard ORM, since its use case, behavior and implementation diagrams model aspects beyond static structures. Such additional modeling capabilities of UML and ORM extensions are beyond the scope of this article, which focuses on the conceptual data perspective. For this perspective, ORM diagrams are graphically more expressive than UML class diagrams.

Moreover, ORM diagrams may be used in conjunction with the other UML diagrams, and may even be transformed into UML class diagrams.

The clarity of a language is a measure of how easy it is to understand and use. To begin with, the language should be unambiguous. Ideally, the meaning of diagrams or textual expressions in the language should be intuitively obvious. At a minimum, the language concepts and notations should be easily learnt and remembered.

Semantic stability is a measure of how well models or queries expressed in the language retain their original intent in the face of changes to the application. The more changes one is forced to make to a model or query to cope with an application change, the less stable it is.

Semantic relevance requires that only conceptually relevant details need be modeled. Any aspect irrelevant to the meaning (e.g. implementation choices, machine efficiency) should be avoided. This is called the conceptualization principle [9]. Validation mechanisms are ways in which domain experts can check whether the model matches the application. For example, static features may be checked by verbalization and multiple instantiation, and dynamic features may be checked by simulation.

Abstraction mechanisms are ways in which unwanted details may be removed from immediate consideration. This is especially important with large models. ORM diagrams tend to be more detailed and take up more space than corresponding UML models, so abstraction mechanisms are often used. Various mechanisms such as modularization, refinement levels, feature toggles, layering, and object zoom can be used to hide and show just that part of the model relevant to a user's immediate needs. With minor variations, these techniques can be applied to both ORM and UML. ORM also includes an attribute abstraction procedure that can be adapted to generate a UML or ER diagram as a view.

A formal foundation ensures models are unambiguous and executable (e.g. to automate the storage, verification, transformation and simulation of models). One particular benefit is to allow formal proofs of equivalence and implication between alternative models for the same application [8]. Although ORM's richer graphic constraint notation provides a more complete diagrammatic treatment of schema transformations, use of textual constraint languages can partly offset this advantage.

With respect to their data modeling constructs, both UML and ORM have an adequate formal foundation.

Since the ORM and UML languages are roughly comparable with regard to abstraction mechanisms and formal foundations, our comparison focuses on the criteria of expressibility, clarity, stability, relevance and validation.

4.2 Multi-valued attributes

Suppose that we are interested in recording the names of employees, as well as the sports they play (if any). In ORM, we might model this situation as shown in Figure 7a. The mandatory role dot indicates that each employee has a name. The absence of mandatory role dot on the first role of the Plays fact type indicates that this role is optional (it is possible that some employee plays no sport). The lack of a mandatory role dot on the roles of EmpName and Sport does not imply that these roles are optional. If in the global schema an object type has only one fact role, this is implied to be mandatory unless the object type has been declared independent. So if EmpName and Sport have no other roles in the complete application, their roles shown here are implicitly mandatory.

This is of little importance, since implied constraints are automatically enforced with no additional overhead.

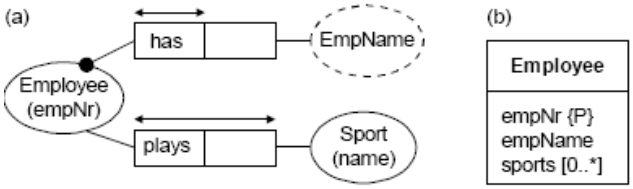


Fig.7 Plays depicted as an ORM m:n fact type (a) and a UML multi-valued attribute (b)

Since an employee may play many sports, and a sport may be played by many employees, Plays is a many-to-many (m:n) relationship type. This is shown in ORM by making the uniqueness constraint span both roles. Visually, this indicates that for each population of the fact type, only the combination of values for the two roles needs to be unique. In other words, each employee-sport pair can occur on at most one row of the associated fact table. Since it is understood that the population of any fact type is a set of rows (not a bag of rows), such a spanning uniqueness constraint always applies. We only show this constraint if no stronger one exists. For example, the uniqueness constraint on the empname fact type is stronger, since it spans just one role; so we don't bother adding the weaker, 2-role uniqueness constraint. Read from left to right, the empname relationship type is many-to-one (n:1), since employees have at most one name, but the same name may refer to many employees.

5 Conclusions

As we used other methodologies for modeling data at the conceptual level, such as UML or ER, over the time in different projects, we consider that the

ORM diagram are more graphically expressive than ER or UML class diagrams.
We've barely scratched the surface of UML or ORM, but many of the fundamentals have been introduced. In later issues, we'll compare UML associations with ORM predicates, fact tables with object diagrams, UML multiplicity constraints with ORM mandatory and frequency (including uniqueness) constraints, UML association classes with ORM nesting, and UML qualified associations with ORM co-referencing.
We'll also discuss more advanced constraints, aggregation, subtyping, derivation rules and queries.
To finalize this article, to give you a real example of applying ORM in data modeling we made an application which contains this facility. It's about a web portal by which a company presents online the projects it developed over the time for different customers. We used the ORM methodology to create the structure of the relational database used by the portal to store projects information.

References:

[1] Bloesch, A.C. & Halpin, T.A. 1996, 'ConQuer: a conceptual query language', Proc. ER'96: 15th Int. Conf. on conceptual modeling, Springer LNCS, no. 1157, pp. 121-33.
[2] Bloesch, A.C. & Halpin, T.A. 1997, 'Conceptual queries using ConQuer-II', Proc. ER'97: 16th Int. Conf. on conceptual modeling, Springer LNCS, no. 1331, pp. 113-26 (online at www.orm.net).
[3] Halpin, T.A. 1998, 'Conceptual Queries', Database Newsletter, vol. 26, no. 2, ed. R.G. Ross, Database Research Group, Inc., Boston MA (March/April 1998) (online at www.orm.net).
[4] Halpin, T.A. 2001a, Information Modeling and relational Databases, Morgan Kaufmann Publishers, San Francisco (www.mkp.com/books_catalog/catalog.asp?ISBN=1-55860-672-6).
[5] Blaha, M. & Premerlani, W. 1998, Object-Oriented Modeling and Design for Database Applications, Prentice Hall, New Jersey.
[6] Fowler, M. with Scott, K. 1997, UML Distilled, Addison-Wesley.
[7] OMG UML Revision Task Force website, http://uml.systemhouse.mci.com/.
[8] UML Partners 1997, UML Semantics, version 1.1, OMG document ad/97-08-04,www.omg.org.
[9] UML Partners 1997, UML Notation Guide, version 1.1, OMG document ad/97-08-05, www.omg.org.
[10] Kyounghwan An, Juwan Kim, Processing Location Stream in Moving Object Database, WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATION, Issue 1, Volume 3, January 2006, pg 147-153, ISSN: 1790-0832
[11] Sorapak Pukdesree, Anon Sukstrienwong, Vitalwonhyo Lacharoj, Evaluating of Distributed Database on PC Cluster Computers, WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATION, Issue 10, Volume 3, October 2006, pg. 1863-1868, ISSN: 1709-0832
[12] Tassawar Iqbal, Nadeem Daudpota, XML based Framework for ETL Processes for Relational Databases, WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATION, Issue 7, Volume 3, July 2006, pg.1402-1406, ISSN: 1709-0832