ORM diagrams intro

Max Talanov.

February 25, 2007.

Contents

1 Intro

Have you ever asked yourself a question, what should be designed first: UML diagram or DSD(Data structure diagram)? And why you choose this way, only because your favorite framework or methodology suggest to do this way,

or why it is necessary to generate not optimal DB structure or model classes? Why we still create ORM in the implementation stage though it should be created in design stage?

2 Problem

In a company I work, for some reason, we have used AppFuse as a base for web-development and DB centric approach.

1. Create DSD
2. Generate DB according to DSD
3. Generate ORM and model classes via MiddleGen

This approach made a good job for us. But, some day we have tried Grails (Rails for Groovy) web-development. The Rails development implies different ORM approach, which I called, model centric:

1. Create class diagram (model classes)
2. Generate model
3. Add mapping meta definitions
4. Generate DB

To be honest, we often skipped first two stages.
Rails development approach was very attractive, but it's pluses (coding by convention) could be considered as minuses. When I came to our DB architect with new ideas of new web-development approach, he told me: "Well, this is no good because this coding convention is not compatible with our coding convention and this is bad because we can not generate as good DB as it should be in our project".
About our project: it is not simple site, but web-based document flow.
Then it was my turn to disagree: "So, you suggest to generate model, following DB centric approach, but you know that this is not the best way out, because generated model is not so good as it should be". After some minutes of discussion, we have decided that we need some tool for ORM modeling. That was the beginning of the ORM diagrams.

3 Solution

First of all I have started to digging in the Hibernate, this mapping was well developed, but some times seems too complex. I have found a relief and that was the ActiveRecord of Ruby. Sometimes it seemed to me that it clarified obscure moments of Hibernate. The idea was to create clear and flexible diagram representation of ORM with key options inherited from Hibernate and ActiveRecord.

3.1 Indicators

When I draw DSD tables and UML classes in one diagram and tried to add some mapping tables, I saw that I needed to distinguish one from an other unambiguously. The solution was quite simple, indicators:

(c) for class

(m) for mapping

(t) for table

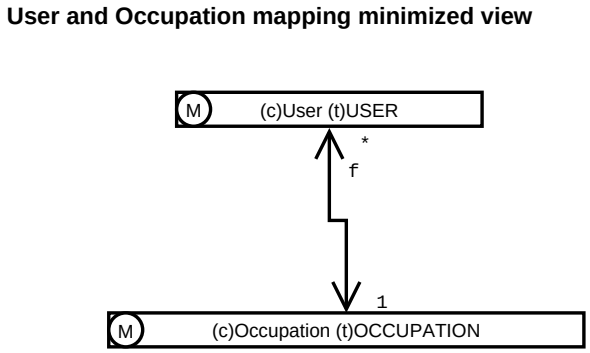

Figure 1: Mapping minimized view

On Figure 1 is only one type of indicator: mapping indicator.
This diagram represents minimized mapping centric view, there are 2 mapping tables, one association one to many with foreign key in *USER* table.
Names of mapping tables are formed by the rule *(c)ClassName (t)TABLENAME*.

3.2 Elements

Then I have tried to combine main mapping construction elements of Hibernate and ActiveRecord.

<**id**> identification property declaration (corresponding primary key)

<**p**> property declaration, could be omitted

<**scaffold**> scaffold declaration

I have used id and property elements from Hibernate mapping and added scaffold from Active record. This way, I tried to use properties with and without scaffolding in the same class. (If scaffold is used all properties that were not specified explicitly, use scaffold in case of ActiveRecord or generated according to naming convention in case of Hibernate.

On Figure 2 are four mappings tables. Elements and associations are in <> followed by property name.

3.3 Associations

Inspired by Active record and Hibernate XDoclets, I have used Hibernate association names with ActiveRecord association modifiers. I have used the same notation for associations and elements, because constructs do not interfere, though semantics is completely different.

<**121**> one to one association

<**12m**> one to many association

<**m21**> many to one association

<**m2m**> many to many association

Main idea of association declaration, was to make it strait-forward, simple and unambiguous. ActiveRecord notation was quite close to it. Something like: "map this property of this class to that class through this table with this foreign key pointing to our class and that foreign key pointing to associated class".
Ex.: (c)Occupation (t)OCCUPATION mapping: one to many association of the property users of the class User, using foreign key U_OID of default table (OCCUPATION).

3.4 References

References play important role in mapping description. Mapping is a description of relationship between entities expressed with references.

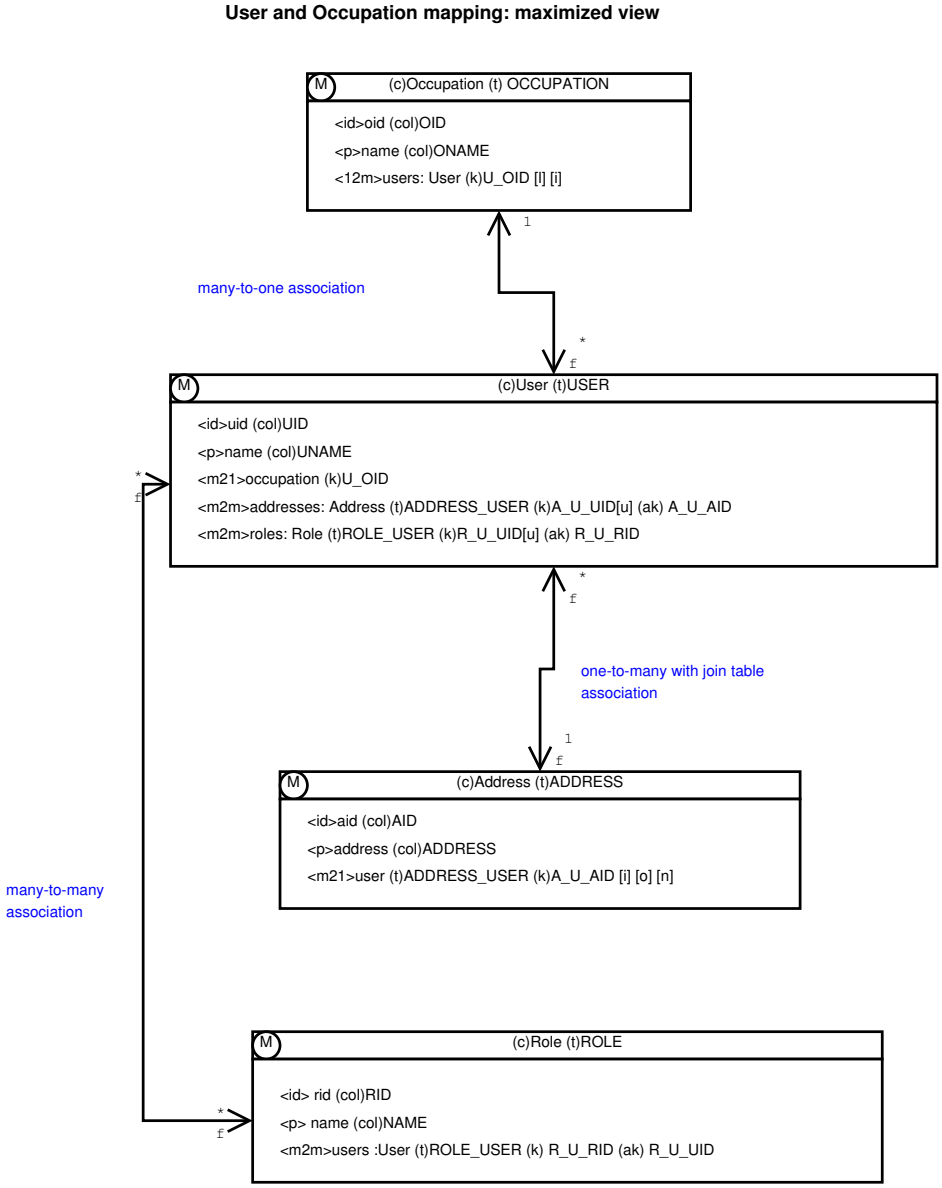

Figure 2: Mapping maximized view

I have used the same syntax for references and indicators, for obvious reasons. Mapping names are constructed as a combination of two references: class and table.
This is references unexhaustive list:

(c) class

(t) table

(k) key (primary or foreign)

(ak) association key (foreign)

(col) column, used in property mapping definition

3.5 Flags

Flags are simple Boolean modifiers of associations and elements, for example:

i inverse

l lazy

o optional

n not null

Flags usage depends on notation: Hibernate or Active record. Flags above are Hibernate flags: inverse to define one end of bidirectional association, lazy to define lazy association, optional to define optional property, not null to define property that can not contain null value.

3.6 Views

There are three kind of views:

- Complete view
- Class centric view
- Mapping centric view

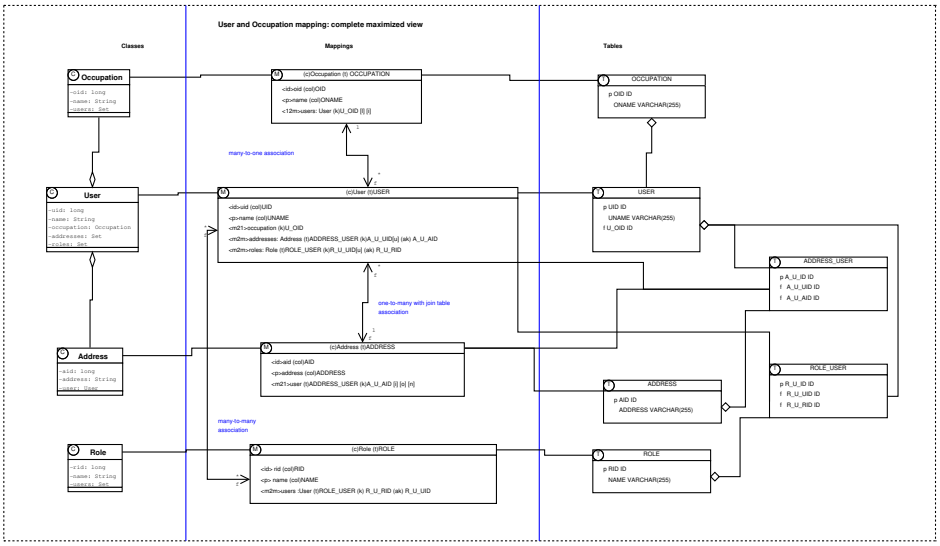

Figure 3: Complete view.

3.6.1 Complete view

Consists of UML classes, DSD and mapping parts, see Figure 3. Each part could be separated by swim line, like in SysML. Class-table mappings are shown as simple lines, associations are shown as arrows, bidirectional or unidirectional. Mapping diagram consist of mapping tables with complete description of properties and associations. This view is unambiguous, but seem to be redundant and overloaded, especially in case of big projects diagrams.
On Figure 3 are: 4 mappings with three types of associations between them.

(c)Occupation (t)OCCUPATION is class-table mapping with: id "oid" property set to column OID (primary key of OCCUPATION table, property "name" set to column ONAME, one to many association with class User is set as property "users" with foreign key U_OID of default table USER, association line shows f near (c)User (t)USER mapping, that contains foreign key. All rest class-table mappings are formed in the same way.

(m)Occupation (m)User is many to one association, uses property Occupation.users in one to many association with class User and property User.occupation with foreign key USER.U_OID in many to one association with Occupation class. This mapping is shown as a bidirectional arrowhead association line.

(m)User (m)Address is one to many association with join table ADDRESS_USER, uses property User.addresses with foreign key ADDRESS_USER.A_U_UID which is unique, association key ADDRESS_USER.A_U_AID in many to many association with Address class and property Address.user with foreign key ADDRESS_USER.A_U_AID, it is optional, nullable and inverse in many to one association with class User.

(m)User (m)Role is many to many association with join table ROLE_USER, uses property User.roles with foreign key ROLE_USER.R_U_UID, which is unique, association key ROLE_USER.R_U_RID in many to many association with class Role and property Role.users with foreign key ROLE_USER.R_U_RID and association key ROLE_USER.R_U_UID in many to many association with class User.

This is exhaustive, but heavy loaded view. It contains complete description of mappings in special mapping tables, it is quite tightly linked to Hibernate syntax with some ActiveRecord extensions.

3.6.2 Class centric view

Consist of UML class diagram, mapping lines with description and DSD.
I have tried to create new kind of view not tigtly linked to Hibernate or ActiveRecord notation. I have chosen classes as a base for mapping description, because both Hibernate and ActiveRecord are class centric, so I have tried to create notation as close as possible to UML. See Figure 4. I have added mapping constructs to class descriptions.

Main changes, in comparison to Complete View on Figure 2 are: I have changed <scaffold> to <[s]> element-flag, associations arrows changed to aggregation with additional description of properties and foreign keys of association. I have used *(M)* indicator to create many-to-many association with intermediate table.
Mainly, mapping is based on lines in this diagram.

class-table shown by plan line

class-class is shown by aggregation line, labeled with multiplication, class property name and foreign keys names in case of join table mapping

On Figure 4 are four classes, four class-table mappings and three class-class associations.

(c)Occupation (t)OCCUPATION mapping contains one property with scaffold and one without, name is set to ONAME column.

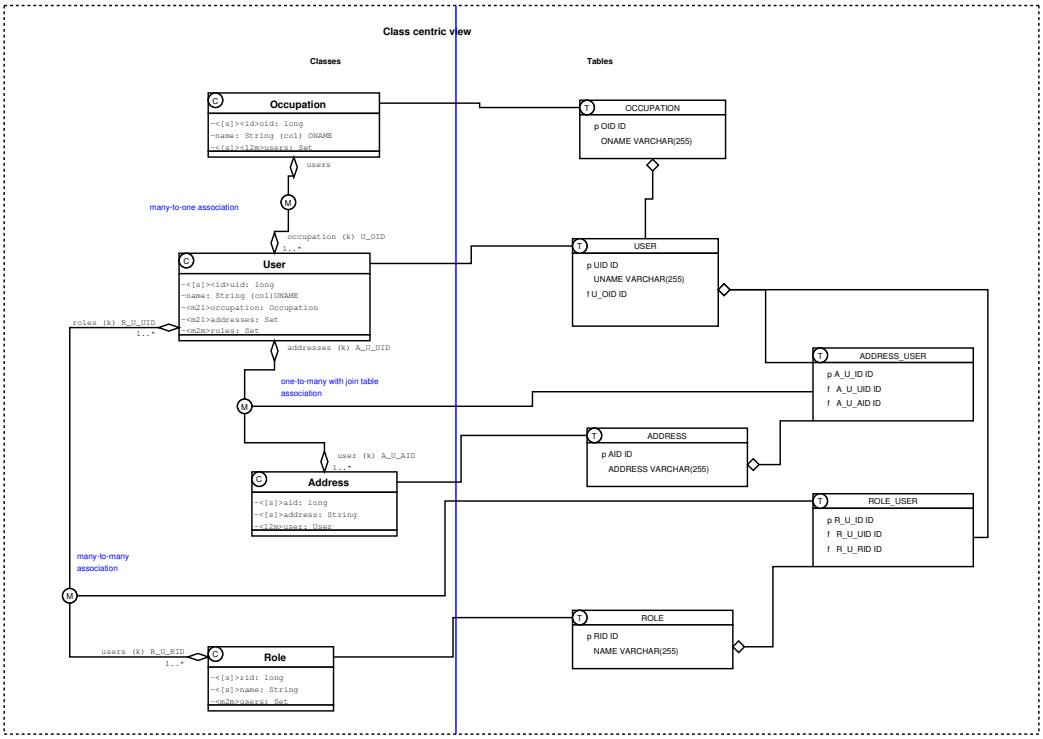

Figure 4: **Class centric view.**

(c)Occupation (c)User many to one association contains property Occupation.users of type Set in one to many association, property User.occupation of type Occupation in many to one association with foreign key U_OID of default table of User class mapping, USER.

(c) User (c)Address many to one association with join table ADDRESS_USER, contains property User.addresses of type Set with corresponding foreign key ADDRESS_USER.A_U_UID in many to one association and property Address.user of type User with corresponding foreign key ADDRESS_USER.A_U_AID in one to many association.

(c)User (c)Role many to many association with join table ROLE_USER, contains property User.roles of type Set with corresponding foreign key ROLE_USER.R_U_UID in many to many association and property Role.users of type Set with corresponding foreign key ROLE_USER.R_U_RID in many to many association.

This view stands little bit aside of two other views, but is quite complete and exhaustive and less redundant than others.

3.6.3 Mapping centric view

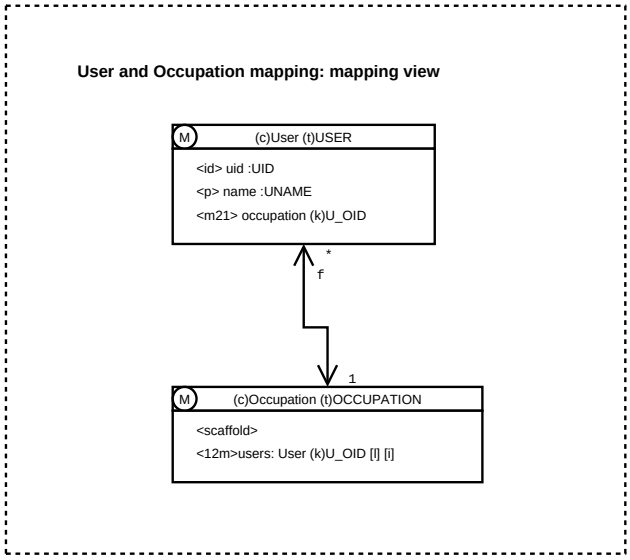

Figure 5: **Mapping centric view.**

This view consist of mapping diagram (a-la MiddleGen) only. See Figure 5, uses mapping description of the complete view, but no classes and

tables included in it. This view is good for big projects with overloaded diagrams.

4 Tools

There are a lot of really good tools for creating DSD and UML diagrams, but I have found only one tool for complex solution for both of diagrams types, see ORM diagram solution of Visual paradigm. I suppose this not enough and their approach is too strait forward.
All diagrams above have been created with Dia, it is real good but has several limitations, and not really UML or DSD tool, that's why I had to use (M) indicator in all class-class associations in Class centric view, see Figure 4, Dia does not support bidirectional aggregation. I have tried to create ORM like diagram in ArgoUML, see Figure 6. Resulting diagram is not so good as it should be, because some limitations based on UML notation, but seems to me that it could be extended to support DSD and ORM diagrams.

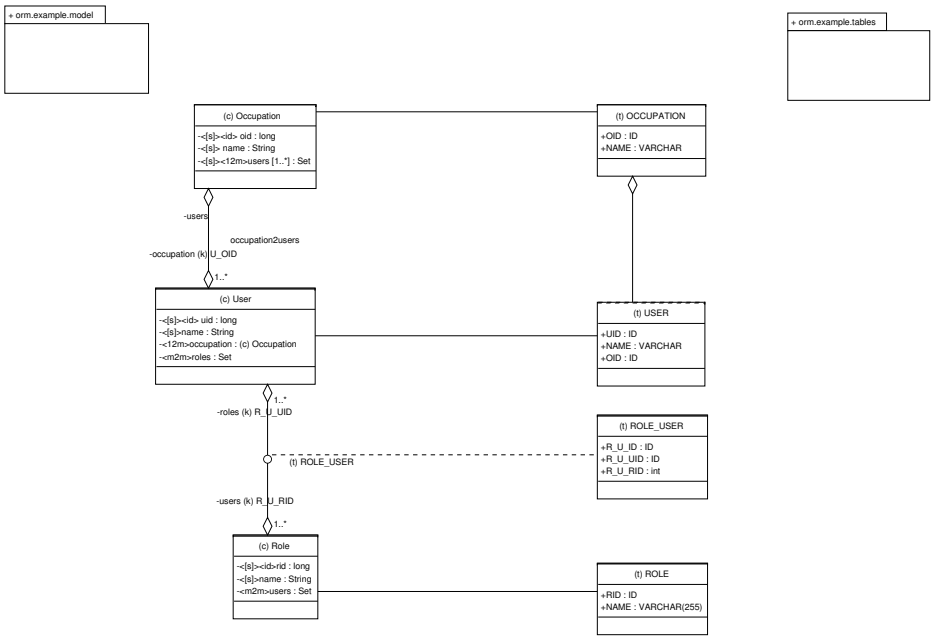

Figure 6: ORM generated with ArgoUML

5 Conclusion

Does not matter which way we use: draw UML class diagrams and then create DSDs or draw DSDs first, we create ORMs during implementation stage, though ORMs play important role in modern projects and should be developed during design stage. Even more, generating model classes and ORMs from DB schema (MiddleGen way) or generating DB tables from model classes (ActiveRecord way) has several minuses: not so good DB structure or not so good model classes structure. We can work it out, creating model classes diagram and DSD, but mappings are still in implementation stage. That's why, I have tried to create some kind of notation suitable for mapping description and produced three views. Most interesting is Class centric view not so overloaded and based on UML notation, see Figure 4.
This is first stage, but still useful, ORM diagram is helpful during creation HQL requests, second stage should be creation of the proper tool for complete UML-DSD-ORM design. It seems that open source Java based ArgoUML UML modelling tool could be good starting point to test the concepts of the complex UML-DSD-ORM diagrams solution.

6 Thesaurus

This section contains several definitions from Wikipedia.

ORM Object-Relational mapping (aka O/RM, ORM, and O/R mapping), is a programming technique for converting data between incompatible type systems in databases and Object-oriented programming languages. In effect, this creates a "virtual object database" which can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to create their own ORM tools. Wikipedia address.

DSD A data structure diagram (DSD) is a data model or diagram used to describe conceptual data models by providing graphical notations which document entities and their relationships, and the constraints that binds them. The basic graphic elements of DSDs are boxes, representing entities, and arrows, representing relationships. Data structure diagrams are most useful for documenting complex data entities. Wikipedia address.

7 See also

1. NHibernate site
2. ActiveRecord for .NET site
3. ActiveRecord for Ruby on rails
4. Hibernate site
5. ArgoUML site
6. ORM diagram implementation of Visual paradigm company