Radboud University Nijmegen

Master Thesis Information Science

Combining Probabilistic Relational Models (PRM) with Object Role Models (ORM)

Thesis Number: 154 IK

Author
Koen Hulsman

Supervisor
Dr. Patrick van Bommel

July 6, 2011

Contents

Chapter 1

Introduction

1.1 Problem Statement

For over 10 years, the idea of Probabilistic Relational Models is floating around in scientific IT-research. Already in 1997, [KP97] described a language to represent Object Oriented Bayesian Networks, another name under which PRMs are known. Still, here we are in 2011 and a lot of problems with these PRMs haven't been tackled sufficiently.

Before getting into the problems with PRMs, let us first look at the main idea behind PRMs. When modeling a complex domain, uncertainty is one of the main issues to tackle. Of course, when problems arise, so do solutions: in this case Bayesian Networks are most common to use. However, BNs give us a major problem: they are not that good in modeling. In fact, [KP97] describe it as: *the task of programming using logical circuits.*

So we need something to do the modeling part. [KP97] decided to use Object Orientation for this. They name it Object Oriented Bayesian Networks (OOBN), because it is a combination of Object Orientation and Bayesian Networks. In the last decade both the terms OOBN and PRM have been used for the same thing, with PRM eventually being the most commonly used one. According to [GFK+07] a PRM specifies how probabilities can be distributed over a database. Their article then specifies a PRM over their database model. However, their database model is very rudimentary, giving a big opportunity for improvement of this model.

In 2010, [SEJ10] try to make a practical implementation of a PRM, connecting it to UML CD. With this, they improve the database model [GFK+07] give, but it still has its limitations: UML is not able to express a lot of constraints and their article is limited to a security domain.

After digging into the research behind Probabilistic Relational Modeling, we see a recurrent problem: finding a good graphical modeling language on which a PRM can be expressed, which is able to express constraints on the model itself. In this thesis it is tried to use an existing modeling language, ORM, and take a look if it can support PRMs without having the problem of being graphically limited. So, can ORM support the graphical representation of a PRM? And if it can not, is it easily extended to do so?

1.2 Research Question & Method

The main goal of this thesis is to look at existing PRM frameworks and try to combine PRM with the modeling language ORM. The focus lies on graphical representations, not the formal part of models. This means that we will not try to validate or formal define the examples and models we give. It is a study of what is graphically possible with PRM models and its combination with ORM. To research the graphical representation of PRM in ORM the following main research question has been defined:

How can Probabilistic Relational Models be graphically expressed in ORM and what can be the possibilities of this expression?

To answer this question, a couple of subquestions need to be answered. The basis of subquestion 1, 2 and 3 is the scientific literature. All facts and figures are based upon the found literature. Not always references to the articles will be present, but unless stated otherwise, it are the findings from the articles written down in our own words. We are, as always, standing on the shoulders of giants.

1.2.1 What is a Probabilistic Relational Model (PRM)?

A quite fundamental subtopic is to explain what a PRM exactly is. Which models qualify for this term? Where did the PRMs originate from? We also take a look at the differences between PRMs and OOBNs, two terms used for what we now call PRMs. We will use the scientific literature about PRMs and OOBNs as source for answering these question. Examples of this literature are [BB58], [BW00a], [GFK+07], [KP97], [Pfe99], [TWG10]. Output of this subquestion will be a global definition of PRMs, a short overview of the articles written about PRMs and the ideas found in them and an explanation of the terms OOBN and PRM.

1.2.2 What does a PRM Framework look like?

Now we have defined what the term PRM means, it is needed to find examples of graphical representations of PRMs. Again, the literature gives us the information we need. This study is to find inspiration and examples of how to represent the PRMs in other modeling languages and the good and bad practices. This will mean we will compare the three frameworks with each other. Also, we will introduce the case study used to give examples of these graphical representations. We will use three frameworks, by [KP97], [BW00a] and [GFK+07]. The output of this chapter will be a description of our case study and descriptions of three frameworks, one which will be used in the creation of a connection between PRMs and ORM.

1.2.3 What is Object Role Modeling (ORM)?

In this subquestion the existing modeling language, ORM, will be described and we will discuss why ORM (and not another from all the available languages) is used. This subquestion is not only for the people who do not know ORM. The goal of this subquestion is also to formulate an interpretation and semantics of the language and to make a good foundation for the usage of ORM in the last subquestion. We will use [Hal98] as source for the ORM description. This article is written by Terry Halpin, one the founding fathers of ORM. The case study will be used to show the construction of an ORM schema. Finally we will discuss why

we used ORM and not an other language, like UML, for this thesis. With this subquestion we lay the last foundation for converting ORM to a PRM-supporting language.

1.2.4 ORM and PRM: a Working Combination with new Possibilities?

With this subquestion we try to convert the found graphical representations from the second subquestion to ORM. We will use the framework of [GFK+07] as source for the conversion. There will be an in depth study of what is possible to express in ORM regarding PRMs and what is not. Besides that, we take a look if the advantages ORM has over other modeling languages also gives an advantage when expressing PRMs in them. The result will be a mapping from the framework of [GFK+07] (GFW) and ORM as described in subquestion 3 and we will discuss the answer to the question why using ORM can have an advantage over using other modeling languages for PRM. We will try to play a bit with the graphical possibilities has and see if they give new options in the combination of ORM and PRM, called P-ORM. The output of this subquestion will be many conversion-figures between GFW and ORM, and figures which illustrate the (new) possibilities of P-ORM.

1.2.5 Answering the research question

After answering the subquestions, we should have found if it is possible to combine PRM with ORM and the new possibilities that this combination may give. The answer on this question will be a summary of the important findings we did in subquestion 4, backed by the theory we found in subquestions 1, 2 and 3. Possible future research will also be discussed.

1.3 Structure of this thesis

The structure of this thesis will be as follows: you are now in chapter one, introducing this thesis. Chapter two will show the results of the research for subquestion 1: *What is a PRM?*. Chapter three will do the same for subquestion 2: *What does a PRM Framework look like?*. Chapter four will describe ORM and the other topics belonging to the subquestion *What is Object Role Modeling (ORM)?*. Chapter five is about subquestion 5: *ORM and PRM: a working combination with new possibilities?*. Chapter six will conclude this thesis with a summary, evaluation of the subquestions, the answer to our main research question and possibilities for future work.

Chapter 2

What is a PRM?

This chapter explains what Probabilistic Relational Models (PRMs) exactly are and why they are used. The first part will give definitions of what a PRM is by walking you through the history of PRMs. The second part summarizes this history and the different definitions of PRMs.

2.1 The core of PRMs: Bayesian Networks

But first we start with a short description of the basis of all PRMs, Bayesian Networks (BN). A BN consists of a given set of random variables (integers), which have a fixed relationship with each other. An example of a Bayesian Network can be found in figure 2.1. Most of the time, a BN uses probabilities as variables and can do some kind of prediction of chance. In the example, it is calculated what caused wet grass with a certain probability. It is also possible to add more variables to it, letting the BN also predict values for these variables. The predicting is based on the probability theory of Thomas Bayes [BB58], hence the name Bayesian Networks.

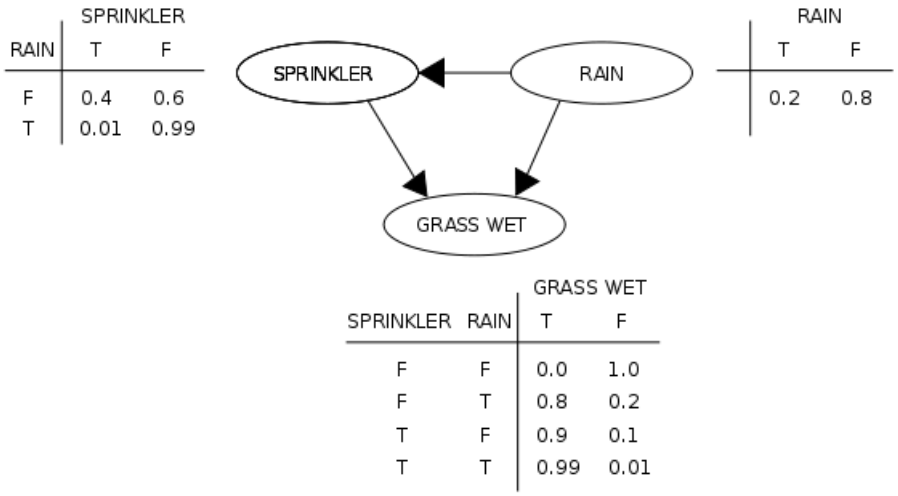


Figure 2.1: Example of a Bayesian Network with probability distribution.

As said, in most cases a BN would be sufficient to model a domain with uncertainty. When modeling, for example, possible diagnosis for certain symptoms (like coughing and shortness for breath are symptoms for pneumonia), a BN will do its job just fine. The chance that a combination of certain symptoms gives away a certain medical condition is (sort of) fixed. However, the given definition of a BN also gives away its major drawbacks: BNs are limited to prespecified variables and fixed relationships [GFK+07].

Besides the above mentioned drawbacks of BNs, there is another one: a BN can only be used in the specific domain it was created for. When creating a model for a complex domain, it would be nice to, with some small editing, be able to use it for another domain. Conclusion: BNs are not suited to be used for a changing and complex domain (a complex domain can be seen as a domain with hard-to-define, changing relationships between entities). So, something else is needed to help us model domains with uncertainty.

2.2 The beginning of PRMs: OOBN

[KP97] seem to have found a suitable solution: combining BNs with Object Orientation (OO). The power of OO lies in the way it can, in a simple manner, express relations between complex and basic objects [KP97]. A complex object, in fact, is nothing more than an object consisting of one or more basic or complex objects. These basic objects can be seen as attributes. Their idea is simple: using BNs to create a probabilistic model using the assignment of values to a certain object. They give it the name OOBN, Object Oriented Bayesian Networks.

The first description of what [KP97] see as an OOBN is found in their introduction and could be summarized as: OOBNs use the declarative probability semantics of BN in combination with the organizational aspects of Object Oriented models. Their OOBN consists of objects, which can be standard random variables (as in traditional BNs) and complexer objects, which can have other objects as attributes. A detailed description of their framework can be found in the next chapter. The objects are then treated as stochastic functions: a function that returns a probability distribution for each value of its inputs. The returned probabilities are distributed via outputs. Complex objects can be calculated by combining different stochastic functions.

To generalize over multiple objects, objects belong to certain classes. Each class has the same probabilistic model underneath it, which gives it the possibility to use it in different contexts. Just like a BN, an OOBN has an unique probability distribution, which gives it the ability to be interpreted unambiguously. This means that if an OOBN contains more than one instantiation, every instantiation has to be defined separately in a BN. Besides generalization, classes also support inheritance. Because each attribute is a stochastic function, a subclass can easily redefine some functions or add new ones.

Besides being a *nice language for representing complex probabilistic models* [KP97], OOBNs also give the opportunity to represent a domain in a hierarchic way, as interconnected objects. [KP97] give a good example themselves, shown in figure 2.2.

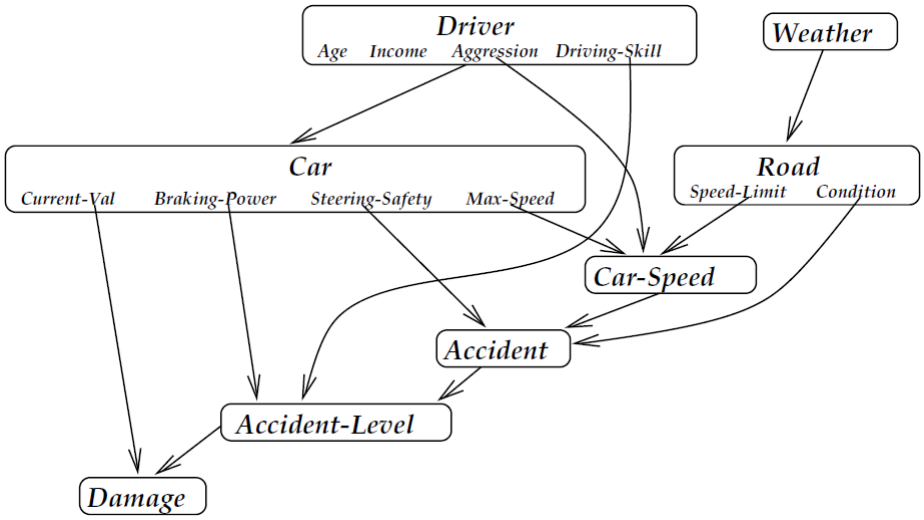


Figure 2.2: Example of hierarchy of interconnected objects [KP97].

2.3 From OOBN to PRM

Using the work of [KP97] as a base, [FGKP99] are the first to formally speak of Probabilistic Relational Models. According to Friedman a PRM allows the properties of an object to probabilistically depend on not only the properties of the object itself, but also on properties of other objects. When giving their formal definition of a PRM, they also state that a PRM can express more information than a BN. This is because a BN can only be expressed over a fixed set of attributes. A PRM, however, can be used over any set of attributes (it can be used over more than one domain). When comparing this to [KP97] the definition is basically the same, stressing the strength of the ability to define a model for more than one context or domain.

2.4 OOBN revisited

The 2000-article by [BW00a] jumps back to [KP97], speaking of OOBN instead of PRM. Directly based on this article is the work by [LB01]. This article is trying to come up with an other solution than [KP97]: they use a top-down approach, where [KP97] had a 'normal' modeling approach. Still, their vision on an OOBN is the same, facilitating the construction and modification of repetitive structures. Also providing a natural way of the reuse of certain model fragments. The biggest benefit of Object Orientation, class hierarchy, is a powerful tool to help construct and modify these models.

Building on the work of [BW00a] and [BW00b], [BFJ03] tries to optimize the practical part of OOBNs. The problem they find with the current implementations of OOBNS is that there has been no attempt to benefit (read: exploit) the instance-structure of OOBNs. This means to do calculations with an OOBN, you first need to extract a BN from it and then create a junction tree from it. To optimize this process, [BFJ03] try to take away the second step, directly translating an OOBN to a junction tree.

2.5 PRM again

The definition [GFK+07] give for PRM, is shown in the conclusion of this chapter. For the first time, a definition of PRM is given which tells us what a PRM contains: a template for a probability distribution over a database which contains (1) a relational component describing the relational schema for the domain and (2) a probabilistic component describing the probabilistic dependencies present in the domain [GFK+07]. When comparing this to the definitions given by [KP97] (1) could be the model consisting of basic- and complex object, while (2) could be the BN, supporting the probability distribution. In the next chapter, the framework of [GFK+07] will be shown in its detail.

2.6 The shortcomings of PRM

PRMs are still not perfect. With their article, [TWG10] try to extend [BW00a]'s framework. Funny thing is that [BW00a] still speak of OOBNs, while [TWG10] speak of PRMs. Comparing definitions of both, it can be said they are the same with PRM being the more popular term for it. Their major problem with existing PRMs is the lack of real usage of class inheritance. The definition of PRMs found in [KP97] and [BW00a] still hold for [TWG10]. But, because they try to fix one of the shortcomings of PRM, an addition to these definitions is done: a PRM, for them, now also contains interfaces, attribute typing, type inheritance and attribute- and reference overloading. The workings of these additions will be explained in the next chapter. For future work, [TWG10] see room for improvement in the graphical section, which is what we are trying to do in this thesis.

2.7 The practical part of PRM

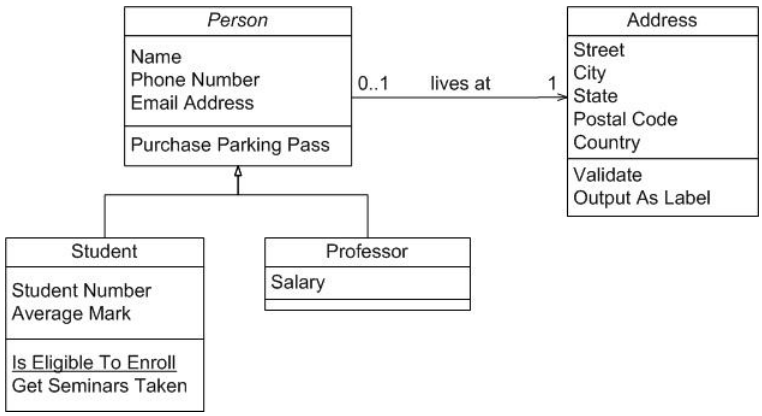


Figure 2.3: Example of a UML Class Diagram (from uml.com.cn)

Being the first article to be all about a practical implementation for PRMs, [SEJ10] do not redefine the PRM or OOBN definitions. [SEJ10]'s definition is similar to the one from [GFK+07]. It describes a PRM containing a template for a probability distribution over an architecture model. The template contains the meta model for the architecture model and

also the probabilistic dependencies between attributes of the classes in the architecture model. This definition can be traced back to [GFK+07] and [BW00a]. An interesting notion from the article is the combination of a PRM with Class Diagrams from the UML language. Because UML Class Diagrams (UML CD, see figure 2.3) and ORM models are related this article showed how PRM can be used in an existing modeling language. Where [GFK+07] still used a limited database model, [SEJ10] use a more powerful language, UML CD. The limitation of this article is that it is specific for a computer security related domain. The main contribution of the article is a package of abstract PRM-classes, applicable on security risks coming from architecture models.

2.8 Concluding

In this chapter, we showed a small time line of the contributions to the PRM and OOBN field, making the reader familiar with the different articles about those two subjects and we tried to show the different definitions these articles gave and compare them to each other. To summarize the what in '*What is a PRM?*' this thesis from now in will use the definition of [GFK+07]: *A PRM specifies a template for a probability distribution over a database. The template includes a relational component that describes the relational schema for our domain, and a probabilistic component that describes the probabilistic dependencies that hold in our domain. A PRM has a coherent formal semantics in terms of probability distributions over sets of relational logic interpretations.*

What exactly is meant by this definition and how it can be expressed, will be shown in the next chapter. Here, three frameworks of PRMs will be discussed, namely [KP97], [BW00a] and [GFK+07].

Chapter 3

What does a PRM Framework look like?

As we saw in chapter 2, there are different implementations of a PRM: first we have the implementations which are still called OOBNs: [KP97], [BW00a]. Then there are the ones who speak of PRMs: [GFK+07], [TWG10] and [SEJ10]. All these different implementations are either based on the framework of [KP97] or on the one given by [BW00a]. While it is strongly based on [KP97] and [BW00a], the framework created by [GFK+07] gives a much more modern view on PRMs. By discussing these three frameworks, a broad understanding about PRMs can be created, giving us the ability to make the combination of ORM and PRM in chapter five.

This chapter will discuss the following subjects: in the second section [KP97]'s framework will be discussed, in the third the frameowkr of [BW00a]. The fourth section will discuss the framework of [GFK+07] and the fifth concludes this chapter. First, a description of the running example used throughout this thesis will be given.

3.1 Running Example

The Running Example below will be used to explain and illustrate concepts we find in the articles about PRM and in the next chapter ORM. The basic notation we use is based on the one [LB01] use for their cow-example. Throughout this and later chapter, the notation may vary. We name this example the Running Example (RE) (as it is based on sports, a fitting name). The RE will slowly develop and be defined using the different perspectives found in the literature. It will be used to illustrate concepts from the articles if the authors themselves fall short in giving a(n) (good) example. If the articles have good examples and figures themselves, we will use them instead of our own Running Example.

The idea behind the RE is the following: a Sporting Team consists of a selection of (1) Players, (2) a Trainer and (3) a Country. (1), (2) determine the Performance of the team. And the Performance determines the Prizes a team will win, the amount of Revenue the team revenues and how large its Fanbase is. The Fanbase is also influenced by the country the team is in. This is because a sport can be favored more in a certain country than in an other. Also, the bigger the country, the more fans there likely will be. The Revenue is also influenced by the

Fanbase. The more fans a team has, the more revenue it makes. The whole schema is shown in figure 3.1.

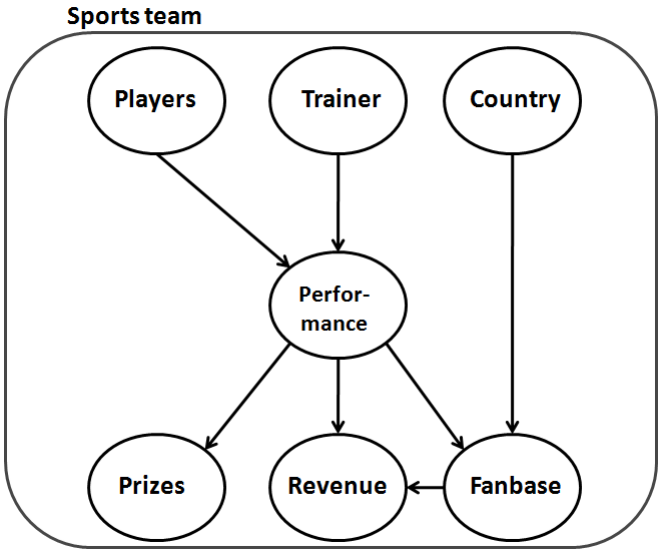


Figure 3.1: Running Example of Sports Team

3.2 OOBN

Before describing the framework [KP97] give, we first need some kind of definition. In their paper, [KP97] describe OOBNs as: *object-oriented Bayesian networks (OOBNs), a powerful and general framework for large-scale knowledge representation using Bayesian networks. As we will show, OOBNs combine clear declarative probabilistic semantics with many of the organizational benefits of an objectoriented framework.* The article by [KP97] is the main source for this section. Unless specified otherwise, all quotations come from their article.

First, a description of [KP97]'s graphical representation will be given. Then, the inheritance in their framework will be defined. Finally, we discuss the strengths and weaknesses of this framework.

3.2.1 Graphical Representation

The basic building block here is the object. This object can refer to a physical entity (a house, a car), an abstract entity (a disease, a chemical substance) or a relationship between different entities.

Definition 1. A *basic type* is a set of values, which is a predefined type (Integers, Booleans, Reals) or a user-defined enumerated set (e.g. week = {monday, tuesday, wednesday,...}). In the RE, Revenue can be seen as predefined, being some kind of integer. Country can be user-defined as country = {Albania, Algeria, ..., Zimbabwe}.

A *basic variable* is a variable which takes values in a basic type. Then there are the structured types, which consist out of basic or structured types.

Definition 2. A *structured type* is a set of values defined by a tuple $(A_1 : t_1 .....A_n : t_n)$, where A_i is the label for the attribute and t_i the corresponding (basic or structured) type. The matching value v_i is that with label A_i and type t_i. In the RE, Players can be seen as a structured type, containing an amount of players of the type Player. The name of the player is the attribute label. A player itself is also a structured type, inheriting attributes from a Person and some additional attributes, belonging to an athlete.

Example 1. In our RE, Trainer is a structured type, containing attributes defining a trainer. These attributes could be Age, Gender, Description and Trainers Experience. Age is a basic predefined type, based on integers (and most of the time < 100). Gender is a basic user-defined type GENDERS $= \{male, female\}$. Trainers Experience is a user-defined type Experience $= \{0, 1-5, 6-10, 10-15, > 15\}$. Description is the description of the appearance of the trainer. It is a structured type, containing attributes like Height, Weight, Eye Color and Hair Color. These are all basic types, with Height and Weight being integers and Eye Color and Hair Color user-defined types.

With these two definitions, objects can be defined. Objects are composed of two types of attributes: *input attributes* and *value attributes*. Input attributes are parameters to the object and influence the choice of values for the vale attributes. *Complex attributes* can be created when these attributes are objects themselves. Value attributes are divided into two types: *output attributes* (visible to rest of model) and *encapsulated attributes* (only visible within the object).

Definition 3. A *simple object* X is of a set of labeled *input attributes* $I_1, ..., I_k$ and a single *output attribute*. All of X's attributes are basic variables.

Definition 4. A *complex object* X is composed of a set of labeled *attributes*. The attributes are partitioned into three sets: the *input attributes* $I(X)$, the *output attributes* $O(X)$ and the *encapsulated attributes* $E(X)$. These two are also called *value attributes* and denoted $A(X)$. The input attributes are (basic or structured) variables.

Example 2. We take Sporting Team (ST) as object from the RE, with the attributes {Players, Trainer, Country, Performance, Fanbase, Revenue, Prizes }. ST qualifies as a *complex object*, because not all of ST's attributes are basic variables (e.g. Trainer is a structured type). ST's *input attributes* are Players, Trainer and Country. The *encapsulated attribute* here is Performance. The output attributes are Fanbase, Revenue and Prizes. The corresponding model is found in figure 3.2, where the dark nodes show input-attributes, light gray nodes show the encapsulated attributes and the white nodes are the output attributes.

With these definitions, we can now define an OOBN. The formal definition for an OOBN is:

Definition 5. An object-oriented Bayesian network consists of a set of class definitions $\mathcal{C}_1, ..., \mathcal{C}_m$ and a single *Situation* object, which has no input attributes, with an associated OONF. A OONF defines the connections between the attributes of a class (see figure 3.4. Besides that, their can be no class recursion: X being of class Y and attribute A of X ($X.A$) also being of class Y.

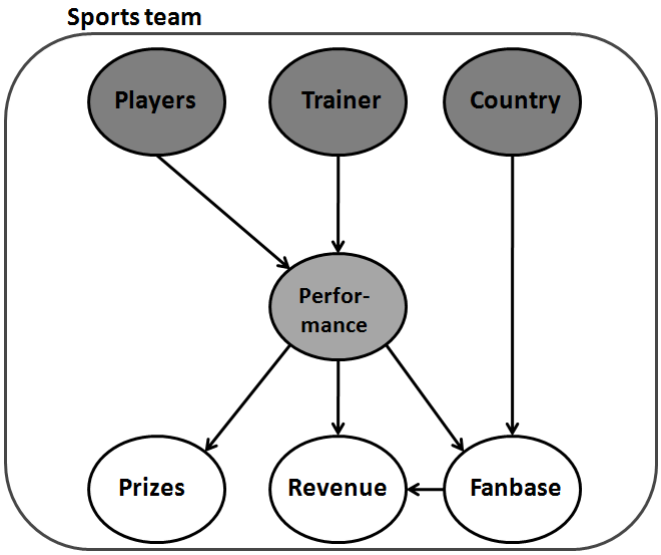


Figure 3.2: ST-Object with corresponding attributes.

In their definition, [KP97] say an OOBN has a single *Situation*-object. However, when studying the text above their definition, it is found that an OOBN in essence is a single situation object, without inputs and with probabilistic properties defined with an OONF (which specifies the conditional distribution of a set of value attributes, with a set of input attributes). A conclusion from this statement is that an OOBN is a situation object. In [BW00a], which use this framework as the foundation for theirs, the Situation-object is ignored. So, it doesn't become really clear from their definitions and examples what a *Situation*-object exactly is and how it is used. The best way to describe it, is saying that with some expansions a situation object can become an OOBN itself. But as [BW00a] ignore the *situation* object, we assume it is not that important.

Example 3. When applying [KP97]'s definitions and examples to our RE, the schemes in figure 3.3 and 3.4 can be constructed. Figure 3.4 is the graphical connection model of the players class. As we can see, it contains one or more objects from the class Player, from which their Physical Strength, Motivation and Speed is used as input to Team Speed, Team Physical Strength and Team Motivation. These three then determine the strength of a team, found in Team Strength. This is the *output-variable* of the class.

When looking at the whole RE, we can construct an OOBN of it. In the OOBN, there are 4 classes (Players, Trainer, Fanbase, Prizes. Players, Trainer, Fanbase and Prizes are instantiations of their corresponding classes, and are complex objects. Country, Performance and Revenue are simple objects. The attributes named under Players, Trainer, Fanbase and Prizes (e.g. Trainer.Age) are the output-types of those objects.

[KP97] continue with expressing the stochastic formulas and probabilistic theory of their framework. Because of the complex level of these definition and this thesis is about graphical representations of PRMs, we will not further discuss them.

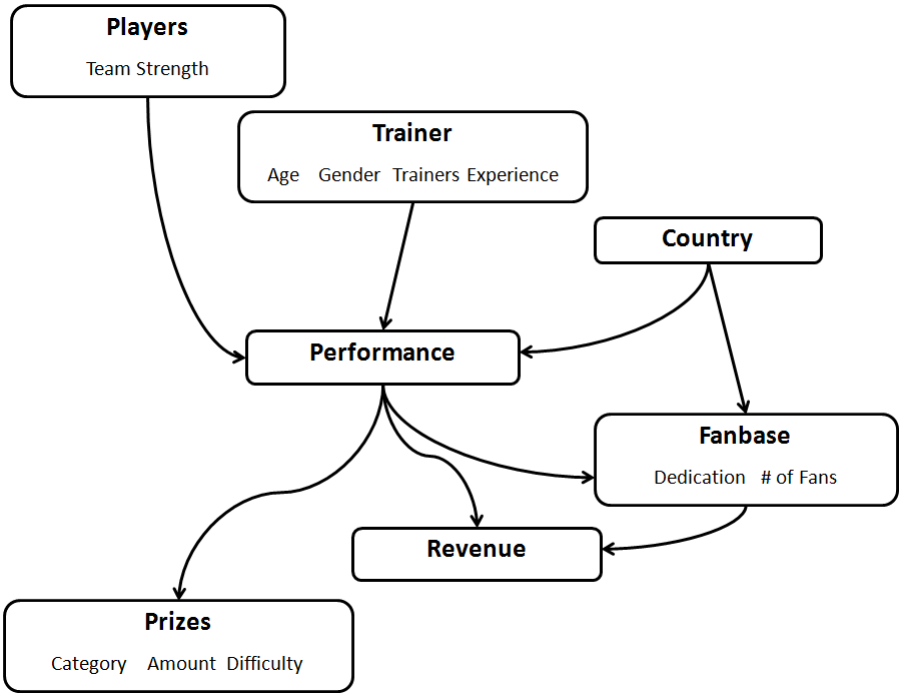


Figure 3.3: OOBN for RE

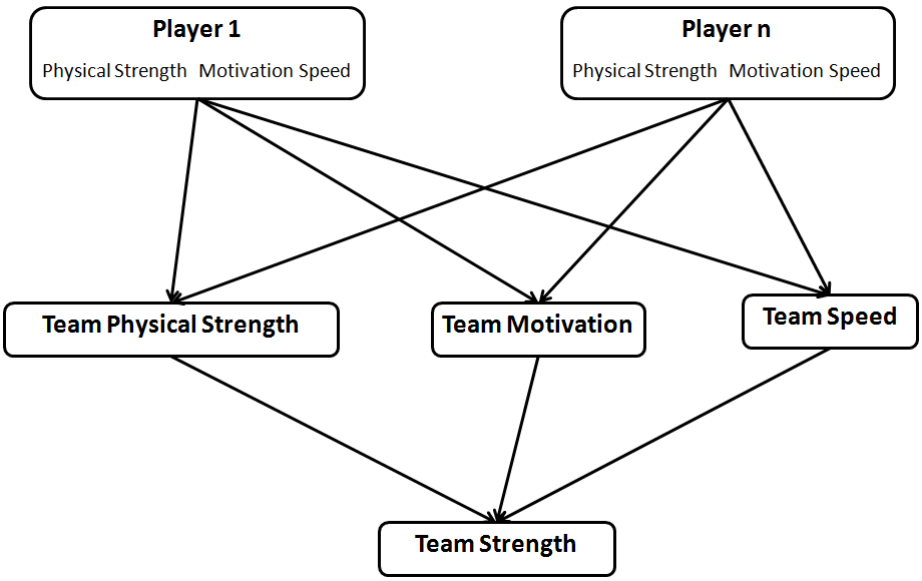


Figure 3.4: Interconnections in Players-Class for RE

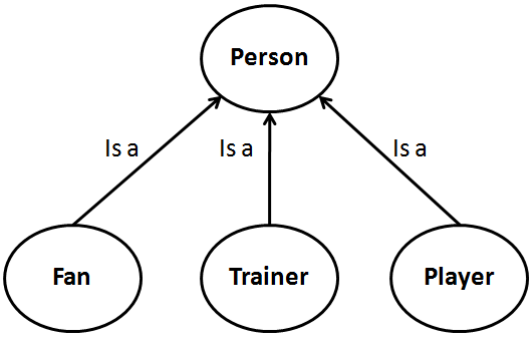


Figure 3.5: *is-a hierarchy* of RE

3.2.2 Class Inheritance in the Framework

Object Orientation is well known for its ability to create subclasses and use inheritance to define the properties of that class. Classes use an *is-a-hierarchy*: if you instantiate a subclass it also belongs to its parent class. With this hierarchy, you are now able to use an instance of a subclass where an instance of the parent class is expected. This means the in- and output of subclasses must at least have the same attributes as its parent-class. To define the inheritance, *interface-types* are used:

Definition 6. $\langle I_1 : t_{I_1}, ..., I_k : t_{I_k} \rightarrow O_1 : t_{O_1}, ..., O_m : t_{O_m} \rangle$, where $\mathcal{I}(C) = \{I_1, ..., I_k\}, \mathcal{O}(C) = \{O_1, ..., O_m\}$ and t_A is the type of attribute A. In short, I is input, O is output and t is the type of that attribute.

A subclass can now be defined as follows:

Definition 7. C' is a subclass of C if $\mathcal{O}(C) \subseteq \mathcal{O}(C')$, and the projected interface type of C' onto $\mathcal{O}(C)$ is a subtype of the interface type of C. The is-a hierarchy of the RE is shown in figure 3.5.

There are two main reasons for definition of the *is-a hierarchy*. The first is that a parent-class can be used as an abstracted version of its subclasses. A subclass is, most of the time, more detailed than its parent-class. An example of this is found in our RE: a Trainer is a Person, but has more attributes (like Experience).

The abstraction property of the is-a hierarchy is also useful for simple objects. An example from the RE: the object Country contains Netherlands, Germany, England, Spain and Italy. But maybe when looking at a sports team, the sport is also played in Russia. We then create a subtype of Country: Country^+ = {the Netherlands, Germany, England, Spain, Italy, Russia}.

As said, there is a second reason to use the is-a hierarchy. Because C' is a subclass of C, we can use all the definitions, types, connections and input mappings of C for C'. It then is only needed to specify the additional attributes of C' with the usual mechanisms: a type declaration, an OONF (see figure 3.4). When dealing with overspecification, [KP97] mention the following: if an attribute A is declared in C' and an attribute by the same name already exists in C, then the C' definition replaces the one in C.

3.2.3 Findings on the Framework

The framework described in their article basically has two components: a relational model, as described in section 3.2.1 and a probabilistic model, where an interconnection model can be found of in figure 3.4. The formal definition of the probabilistic part was not of interest for this thesis, as we focus on graphical representations.

We find two kinds of graphical representations in [KP97]: an object-oriented relational model, which describes the objects (input, output, enhanced) of a model (figure 3.2) and a probabilistic model (figure 3.4), which describes which attributes of a class influence each other. The problem with their framework and its description is that is quite vague about how to give graphical representations of their different models. There are examples in their article, but these are or in words or graphical-without-description. This makes constructing these graphical models a bit like guessing and imitating, basing it on the formal descriptions they give.

Another problem comes forth when looking at [KP97]'s definitions for inheritance. They use a *Situation*-object, but fail in clearly stating what it is an how it works. The best description is that this object is an instantiation of an OOBN.

Because [KP97] do not really define their graphical representations, it is hard to judge them. Their representation of OOBN's (as in figure 3.3) focuses on the interrelation and only gives the output attributes of the classes. They lack a graphical representation of all the attributes belonging to a class and their (inter)relations. The best they can do is a database representation of their own example, shown in figure 3.6, but this model lacks all the possibilities of adding constraints or uncertainty-markers to the model (this is also the case with all the other models in [KP97]'s framework. This is mentioned by themselves as main limitation: *In particular, the language does not allow us to express uncertainty about the identity and number of objects in the model and about the relationships between them. [...] A related restriction is that we cannot express global constraints on a set of objects*.

Then there is the problem of not being able to construct models where the number of objects vary. If we look at the RE, a Player-class in a Sports Team does not always have the same amount of players in a team. A football team has eleven players in the field, while a basketball team has five. This means it needs to be possible to vary the amount of players. It is tried in figure 3.4 to do this with the three dots between the players, but this is of course not the best way to do this.

3.3 Top Down Representation

In their article, [BW00a] take another point of view than [KP97]. The central perspective in this article is the top-down perspective. The description of [BW00a]'s framework will start with its definitions of the class representation and the graphical representation of it. The class hierarchies will be discussed as second. Because [BW00a] state that their framework differs much from the one [KP97] describe, the third section will look at these differences. All citations are from [BW00a] unless specified otherwise.

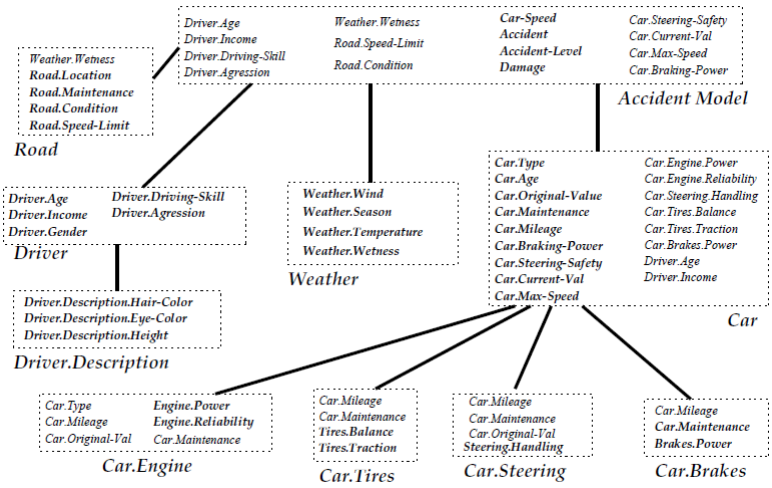


Figure 3.6: Model of database of car accidents-OOBN by [KP97]

3.3.1 Introduction to the Framework

The cornerstone of [BW00a]'s framework is *the class*, as used in a Bayesian Network. The terminology is as follows:

- **Class** A class is a fragment of a Bayesian network. It can contain *instantiations* of other classes.
- **Instantiation** An instantiation is an instantiation of a class within the specification of another class. There can be several instantiations of a single class.

A class contains the building blocks of their framework: *nodes*. A class is viewed as a unit, with two kind of interactions with the outside the class:

- A node inside a class has parents outside the class
- A node outside a class has parents inside the class

[BW00a] see the first type of interaction as problematic. This is because the *inner nodes* can not be changed by nodes outside the class. An inner node can therefor not have a parent outside its own class. They therefor introduce a new kind of node: the *reference node*. It points to a node in another scope and is linked with a *reference link*. An example is found in figure 3.7b. An important limitation to reference nodes is that changes to reference nodes can only be accomplished by changing its *referenced node*. Changing the values of the reference node itself is not possible. The interface of a class is the set of input and output nodes (as they are the nodes visible outside the class).

Classes are pieces of BNs, containing special nodes (instantiation and reference) and special links (reference). Classes have three sets of nodes:

- **Input Nodes** which are all reference nodes

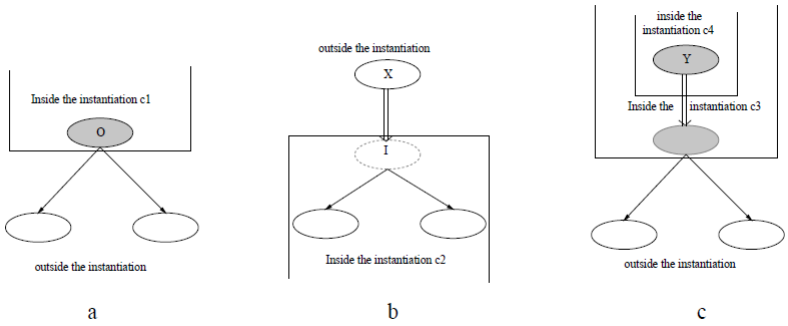


Figure 3.7: Different kinds of input-output interactions in [BW00a].

- **Output Nodes** which can be parents of nodes outsides the class
- **Internal Nodes** which are only visible inside a class or instantiation

The scope of a class consists of all its nodes and all the input and output nodes of the instantiations in that class. The graphical representation of the entities in a class are shown in figure 3.8.

3.3.2 Explaining the Class Representation

A class **T** is a Directed Acyclic Graph (DAG, a graph with no directed loops) over {*I, H, O*}, where *I* are the *input nodes*, *H* the *internal nodes* and *O* the *output nodes*. Because [BW00a] want no recursion in their framework, instantiations of class **T** can not be of type **T**. The following rules apply for **T** and its instantiations *t*:

- *I, H, O* are pairwise disjoint (a node can only be of one type).
- A node of *I* has no parents in *t* (figure 3.9a), no children outside *t* (figure 3.9b) and can have at most one referenced node outside *t* (figure 3.9c).
- A node of *O* has no parents outside *t*.
- A node of *H* has no parents or children outside *t*.

Nodes can be of different kinds:

- **Instantiation** A node representing the instantiation of a class inside another class.
- **Simple Node** a node which is either a:
  - **Reference Node** Specifies input- and output nodes. A reference node can only have one referenced node (the node from which the reference comes from). A reference node has no parents, but can have children. An output node can be a reference node (when an output node of an instantiation is used as an output node of the class containing the instantiation). All input nodes are reference nodes. An internal node can not be a reference node (an internal node may only be accessed from inside the class).

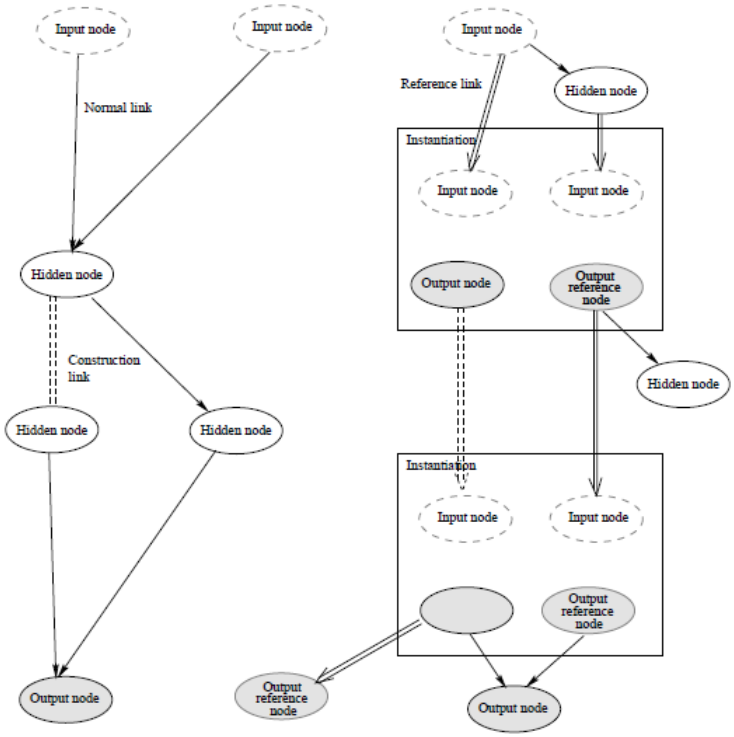


Figure 3.8: The different nodes and links in a class specification [BW00a].

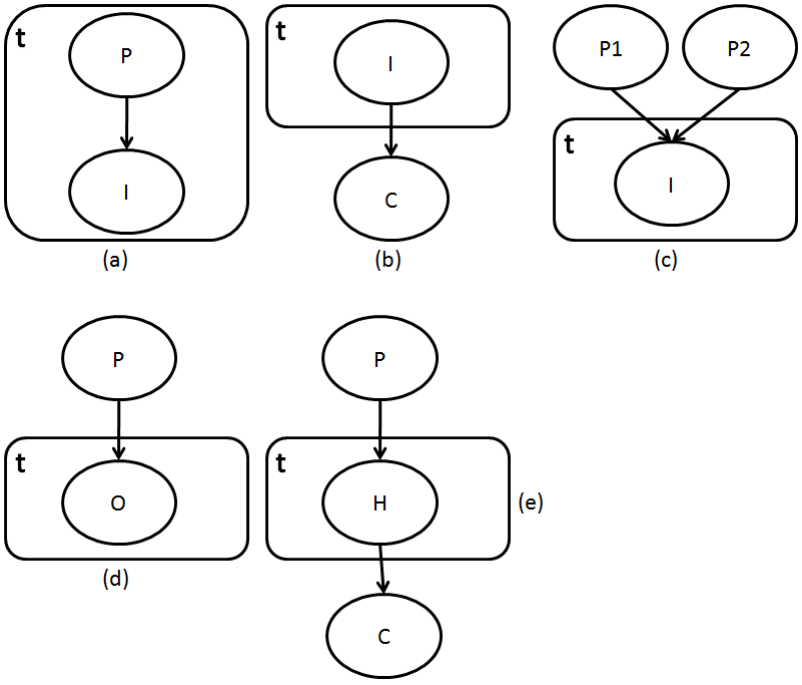


Figure 3.9: Forbidden constructions in the framework of [BW00a].

– **Real Node** a node representing a variable.

Between nodes, links exist. There are three sorts of links: *directed*, *construction* and *reference*. The *directed link* is a normal link, from a *simple node* to a *real node*. *Construction links* are used when specifying a network. They have no impact on the underlying BN, they only specify there is some kind of link between two nodes (they can both be directed or undirected). Then we have the *reference links*, connecting *simple nodes* with *reference nodes*. If we have the connection $A \Rightarrow B$, A is the *referenced node* and B the *reference node*. When looking at the underlying BN, only A is used (as B is only a reference).

From these linkage, a *reference tree* can be built, with at the top the *referenced root*. This is the node to which all the other nodes in the tree are referencing. A class can contain several reference trees. To avoid illegal structures to be specified in the underlying BN, [BW00a] have introduced restrictions, stating that there can not be reference links between two simple nodes in the same class and a referenced node may only have one reference node in the same class. From these restrictions we can conclude the following statements [BW00a]:

- Input nodes can not be used as referenced nodes for simple nodes specified in the same class, but only for input nodes of (other) instantiations. Two input nodes in an instantiation can not have the same referenced node.

- Output nodes can be used as referenced nodes for output nodes in the encapsulating class and for input nodes in a different instantiation in the encapsulated class. Two output nodes of a class can not have the same referenced note.

- Internal nodes can be used as referenced nodes for input nodes of instantiations only. Internal nodes will always be referenced roots if they occur in a reference tree, as they can never be reference nodes.

- A chain of reference links can go in both directions (further inside classes or further out) but once it begins going inside, it can not begin going out again (figure 3.3.2.b).

Figure 3.3.2 gives some examples allowed or disallowed by the restrictions. When specifying the links and these restrictions are not met, changes have to be made to the model. The restricted situations and their valid alternatives are shown in figure 3.11: a) a reference link between two nodes in the same class, b) reference links from a node to nodes in the same scope and c) two reference links to two nodes in the same instantiation.

In their article, [BW00a] discuss the usage of their framework for time slice representation (models that include time). This falls out of the scope of this thesis. However there is an interesting point to this topic: they abruptly dissociate themselves from one of the most constraining rules in BNs: they allow cycles. Because BNs have to be acyclic, this sudden change in approach is to be called weird at the least. [BW00a] give no explanation for the directed cycles, which makes this time-slice example a strange addition to their article. The only mention we find for neglecting their own restriction is the following: *Notice that the reference links [...] are not allowed under the regime of section 3.4, but the restrictions there are relaxed for time slice specification* (where section 3.4 is the section stating the restrictions).

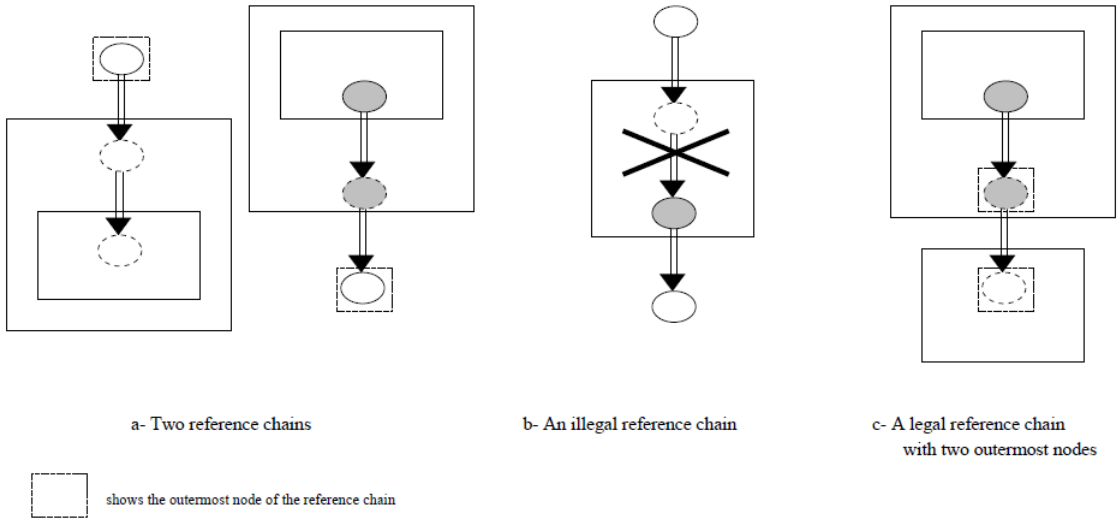


Figure 3.10: [BW00a]: Examples of different kinds of reference chains [BW00a].

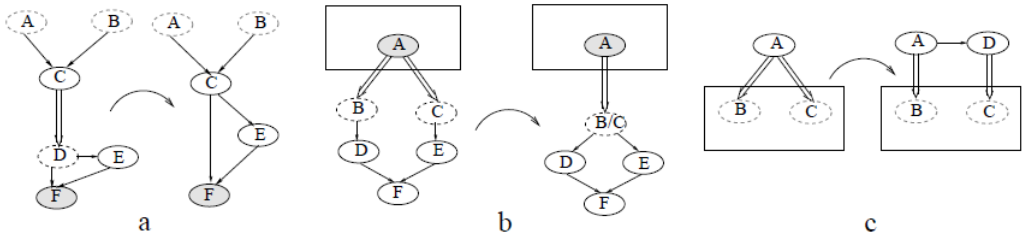


Figure 3.11: Three situations with restricted constructs and their valid alternative [BW00a].

3.3.3 Top-Down Representation and Class Hierarchies

We now come to the part in [BW00a]'s article where class hierarchies are discussed. There is a need for class hierarchy because of the overlap between certain classes. Instead of defining almost the same class twice (with both differing in maybe only two attributes), defining one class with two subclasses is much easier and faster. For example, in our Running Example, a Player and a Trainer are both persons, different in only some attributes. With class hierarchy, both Trainer and Player inherit attributes from the class Person and specify some new attributes for their specific subclass.

In this paragraph the class hierarchy framework found in [BW00a] will be discussed. The goals of their class hierarchy are the same as in Object Oriented Programming (OOP):

- To simplify specifications of similar classes
- To allow automatic updates in classes sharing some properties
- To organize knowledge in a hierarchical way

The second goal is not that relevant for this thesis, but still shows the possible strength of a class hierarchy. The class hierarchy framework of [BW00a] is built on the OOP-idea, borrowing some of their definitions. The first one states the definition of a subclass:

Definition 8. A class **S** over (I_S, H_S, O_S) is a subclass of a class **T** over (I_T, H_T, O_T) (or **T** is a superclass of **S**), denoted $\mathbf{S} \subset \mathbf{T}$ if $I_T \subseteq I_S$, $H_T \subseteq H_S$ and $O_T \subseteq O_S$

So **S** is a subclass of T if **S** at least contains the nodes of **T**. To avoid cycles, **S** can only be a subclass of **T** and not a superclass of **T** at the same time (which could happen if they contain the same set of nodes). [BW00a] also prohibit *multiple inheritance, since an elegant way to do this is yet to be seen.*

An example of class hierarchy in [BW00a]'s framework is found in figure 3.12. Here class **X** has two subclasses: **Y** and **Z**. For both, some redefinition of the probabilities have to be done. For example, instead of P(E|B), we now have P(F|B) and P(E|F) in class **Y**.

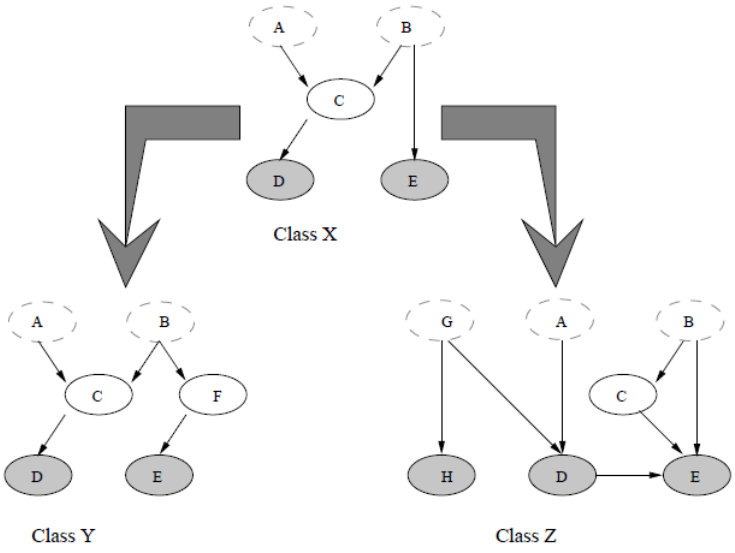


Figure 3.12: Class hierarchy example of [BW00a]

3.3.4 Top-down OOBNs vs. 'Normal' OOBNs

The frameworks from [BW00a] and [KP97] are closely related, but differ in some parts. This section will describe the similarities and differences between the two frameworks. For a better overview of the frameworks, the tables given by [BW00a] are summarized in the table 3.1. Some explanation will be given in the table itself.

The biggest difference between the two frameworks can not be put in the table, because [KP97] have no representation for it: top-down modeling. The reason why there is no top-down modeling in [KP97]'s framework is stated by [BW00a] as follows: *The reason why it is not possible to do top-down modeling in [KP97]'s framework is that all inputs need an annotation and therefore it must be known in advance how an object is to be used.* Because they make use of reference nodes, [BW00a]'s framework does allow top-down modeling.

However, as was the problem with [KP97]'s framework, uncertainty about the amount of objects of a class can not be represented Also, constraints are not available in their framework. Later articles, e.g. [GFK+07] or [TWG10], also do not address these limitations.

3.4 A Modern Approach

In 2007, a more modern view on PRMs was introduced with the article by [GFK+07]. Where [KP97] and [BW00a] stay very close to BNs, [GFK+07] decide to stay closer to a modeling language. Strongly built on the findings of [KP97] and [BW00a], [GFK+07] give a different implementation, much more practical and relational language based.

Despite not stating it explicitly, the graphical representation of their relational model shows many similarities with UML Class Diagrams. Because UML CD and ORM-schemas are very similar, this article is a good starting point for getting to a combined representation of ORM and PRM. This subchapter will first discuss the graphical representation of [GFK+07]s framework. Second, some of the probabilistic theory behind the PRM model will be discussed. As last thing, we will discuss the usage of class hierarchy in the framework. Because the nature of this thesis is not mathematical but graphical oriented, all formulas are just briefly discussed to show some of the theory behind the framework.

3.4.1 Description of the Framework

We already stated in chapter two that the definition [GFK+07] give in their article is the most workable definition of PRMs. For them, a PRM consists of a logical representation of a domain and a probabilistic graphical model template, describing the probability distribution for that representation. This logical representation of the domain and its probability distribution can be graphical, but can also be a formal definition of the domain. [GFK+07] decide to do both with a graphical representation.

In the relational language used to describe a domain, there are classes and attributes. In their example describing the university as a domain, they have the classes Professor, Student, Course and Registration. As with the framework of [KP97], the attributes used are descriptive attributes, giving some import properties of the class. More formally:

Definition 9. A domain has a set of classes $\mathcal{X} = \{X_1, ..., X_n\}$ and a set of descriptive attributes $\mathcal{A}(X)$. Attribute A of class X would be $X.A$. The values of $X.A$ can be found in $\mathcal{V}(X.A)$. So, if we would take the class Student, $\mathcal{A}(\text{Student}) = \{\text{Intelligence, Ranking}\}$. $\mathcal{V}(\text{Student.Intelligence})$ can be $\{high, low\}$

An example of the university domain is found in figure 3.13. As you can see, the structure of the model shows a lot of similarities with the UMLCD showed in figure 2.3. Besides descriptive attributes with a direct value, there also exist *reference slots*. These are to enable the usage of classes in other classes. To formally explain it:

Definition 10. Each class X has reference slots in it, $\mathcal{R}(X)$. One reference slot within X can be denoted as $X.\rho$. A reference slot ρ consists of a domain type Dom[ρ] and a range type Range[ρ]. For example, we look at the reference slot Instructor in the class Course. Dom[Instructor] = Course, Range[Instructor] = Professor. The underlined attributes in figure 3.13-left are the reference slots, connected to its corresponding classes with the dotted lines.

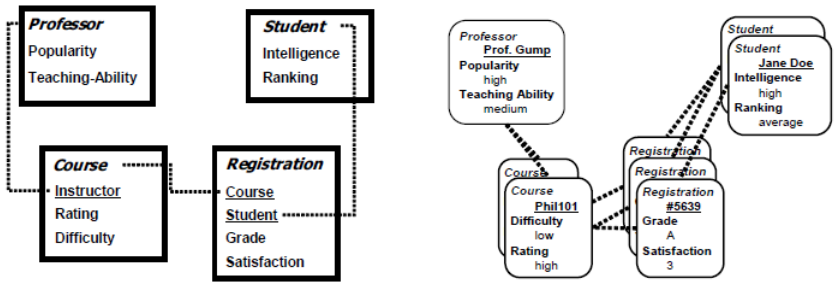


Figure 3.13: Relational schema of the university domain (left) and example population of the schema (right) [GFK+07].

The relational schema can be easily converted to a relational database, with each class being a single table. The descriptive attributes are columns with a value, the reference slots are columns with a reference key (unique key of attribute(s)) to another table with its corresponding class. The connection of the reference slots works both ways, making you able to inverse this relation:

Definition 11. ρ^{-1} describes the inverse function of ρ. Inversing, for example, the InstructedBy reference slot, CourseteachedbyInstructor, we get InstructorteachesCourse. So if we have a reference slot ρ, Dom[ρ] = X, Range[ρ] = Y, Dom[ρ^{-1}] = Y and Range[ρ^{-1}] = X.

The last thing we need to complete the description of the relational models are *slot chains*, which define functions from objects to objects.

Definition 12. A slot chain $\{\rho_1, \ldots, \rho_n\}$ describes a sequence of reference slots, such that for all i Range[ρ_i] = Dom[ρ_{i+1}]. For example Professor.Teaches.Course.Rating gives all the ratings of the courses a Professor teaches.

3.4.2 Instantiating the Relational Model

Without actual values, a PRM wouldn't be of much use. So we need to define instantiations of our objects, describing the population of a relational model. In [GFK+07]'s framework we find the following implementation of an instantiation:

Definition 13. A relational schema has one or more instances, denoted with an $\mathcal{I}$. It specifies for each class X:

1. The set of objects in the class, $\mathcal{I}(X)$
2. The value for each attribute $x.A$
3. The value of each reference slot, denoted as value y for reference slot $x.\rho$

x is an instance of class X, $\mathcal{I}_{x.A}$ is the value of attribute A in $\mathcal{I}$ The right part of figure 3.13 shows an instantiation of the relational schema.

With both the relational model (RM) and a way to describe its population, we can now define a probability distribution over the RM and advance to the PRM.

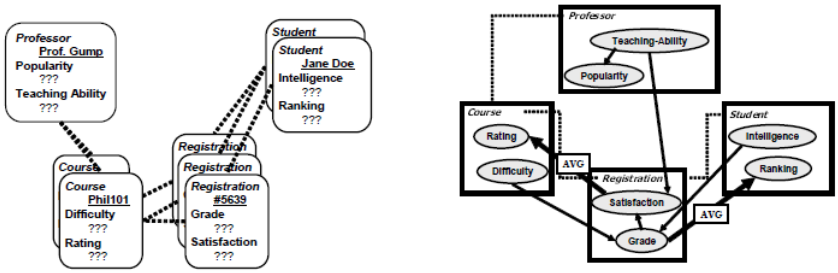


Figure 3.14: Relational skeleton for the university domain (left) and PRM dependency structure of the domain (right) [GFK+07].

3.4.3 Usage of a PRM in the Framework

As we saw in the previous sections, a probabilistic model gives the probability distribution of relational model. The assumption is that the relations in the RM are fixed. The probability distribution can then be given as relations between attributes in the model. A *relational skeleton* is therefor needed.

Definition 14. σ_r is the relational skeleton of a relational schema and can be seen as a partial specification of a schema's instance. In σ the set of objects $\sigma_r(X_i)$ for each class is specified, as are the relations that hold between these objects. Values of attributes are not specified in the relational skeleton.

A PRM can now be depicted in two parts: the PRM dependency structure S and the parameter distribution over it, θ_S. The dependency structure is built out of associations of attributes and their parents. These associations are graphically displayed with arrows in the right schema of figure 3.14. We can now define two kinds of formal parents:

Definition 15. An attribute $X.A$ can depend on $X.B$, giving a dependency for an individual object: for any object x in σ_r, $x.A$ will depend probabilistically on $x.B$. An example from the university domain: a Professor's Popularity depends on the Teaching Ability of the Professor. $X.A$ can also depend on attributes which are related to $X.A$, like $X.\mathbf{K}.B$. $\mathbf{K}$ here is a slot chain. An example (from figure 3.14: a student's grade depends on Registration.Student.Intelligence and Registration.Course.Difficulty.

The dependencies can get quite complex when the amount of attributes attached to a class vary. If we look at the dependency of Student.Ranking on Student.RegisteredIn.Grade, a student isn't enrolled in one class, but in different kinds. An as each student is registered in a different number of courses, these dependencies could get complicated. We can say that $x.A$ is probabilistically depended on the multiset $\{y.B : y \in x.\mathbf{K}\}$. This means: $x.A$ depends on all values of B from different kinds of y in chain $\mathbf{K}$.

To make it possible to do a calculation with a multiset, [GFK+07] introduce *aggregation*, a concept taken from database theory. The probability of $x.A$ will then depend on an aggregation property of the multiset. Examples of aggregations are: mode, mean value, maximum, minimum etc. The Student.Ranking problem can than be solved by using the GPA (Grade Point Average).

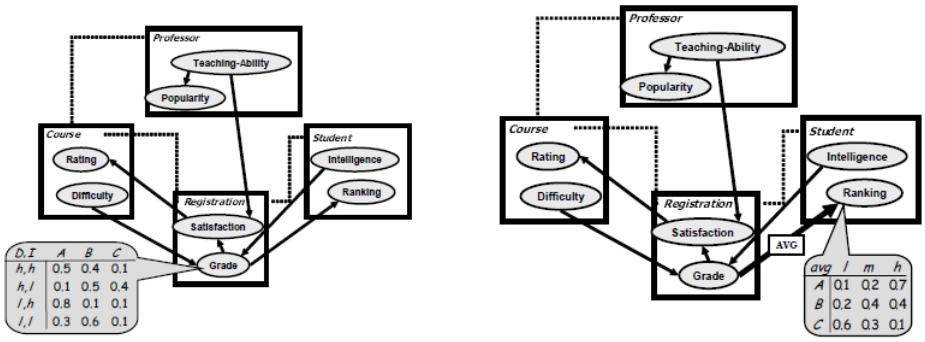


Figure 3.15: CPD for Registration.Grade (left) and CPD for aggregrate dependency of Student.Ranking and Student.RegisteredIn.Grade (right) [GFK+07].

When the PRM dependency structure has been defined, conditional probability distribution (CPD) can be added to the schema. The example from [GFK+07] can be found in figure 3.15.

Definition 16. The parents of $X.A$ are defined as $\text{Pa}(X.A)$. Then, a CPD is added for $X.A$, which specifies $P(X.A \mid \text{Pa}(X.A))$ (The probability $X.A$ has a certain value, given the values of $\text{PA}(X.A)$. One of the central rules in Bayes' probability theory[BB58]). $\mathbf{U}$ is the set of parents of $X.A$, which has some set of values, $\mathcal{V}\mathbf{U}$. For each tuple of values, $\mathbf{u} \in \mathcal{V}(\mathbf{U})$, $P(X.A|\mathbf{u})$ is then the distribution over $(V)(X.A)$. This set of paramaters is then called θ_s.

The formal definition of a PRM Π for a relational schema $\mathcal{R}$ is then as follows [GFK+07]:

Definition 17. For each class $X \in \mathcal{X}$ and each descriptive attribute $A \in \mathcal{A}(X)$:

- a set of parents $\text{Pa}(X.A) = \{U_1, ..., U_n\}$ where each U_i (or parent) has the form $X.B$ or $\gamma(X.\mathbf{K}.B)$, $\mathbf{K}$ being a slot chain and γ an aggregate of $X.\mathbf{K}.B$.
- a legal CPD, $P(X.A \mid \text{Pa}(X.A)))$.

To get the entire probability distribution, factorization is used (this is also done in normal BN). This means: taking the product, over all $x.A$, of the probability in the CPD of the specific value assigned by the instance of the attribute, given the values assigned to its parents [GFK+07]. Formally this would be:

Definition 18. $$P(\mathcal{I} \mid \sigma_r, S, \theta_S) = \prod_{x \in \sigma_r} \prod_{A \in \mathcal{A}(x)} P(\mathcal{I}_{x.A} | \mathcal{I}_{\text{Pa}(x.A)}) = \prod_{X_t} \prod_{A \in \mathcal{A}(X_t)} \prod_{x \in \sigma_r(X_t)} P(\mathcal{I}_{x.A} | \mathcal{I}_{\text{Pa}(x.A)})$$

We will not try to prove definition 18. It is just there to show the Bayesian background of the framework and to show the differences this product-formula has with the Bayesian chain rule. It differs on three aspects:

1. The random variables in this framework are the attributes from a set of objects.
2. Random variables have a varying set of parents, defined by the relational context of the object. In BNs, the set of parents is static.
3. The parameters of the model are shared. The local probability models use the same parameters for attributes of objects in the same class.

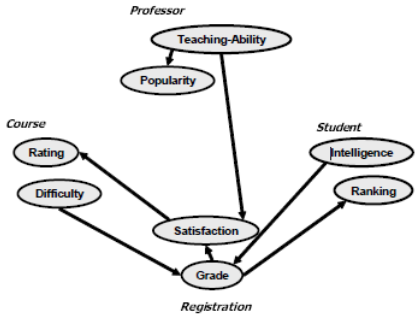


Figure 3.16: Class Dependency Graph for the university domain [GFK+07].

Assuring A-cyclicity

An important property of a BN is that it needs to guarantee a coherent probability distribution (the sum of the probability of all instances needs to be 1). Acyclicity of a graph is what guarantees this ability for BN. As mentioned before, if a PRM would be cyclic, an attribute could be an ancestor of itself or, in other words, the value of an attribute could depend on its own value.

To ensure there are no cycles in a PRM, the process of creating probabilistic dependency edges needs be done very carefully. An example of such a edge is found in the right schema of figure 3.14: the arrow between Registration.Grade and Registration.Satisfaction.

Acyclicity can be guaranteed on *instance* and on *class* level. [GFK+07] do both, but stress that the second one is the most important. To guarantee acyclicity on class level, it is first needed to define edges between attributes [GFK+07]:

Definition 19. G_Π is a *class dependency graph* (see figure 3.16 for a PRM Π. G_Π has a node for each descriptive attribute $X.A$ and contains the follow edges:

1. Type I edges For any attribute $X.A$ and any of its parents $X.B$, an edge from $X.B$ to $X.A$ is introduced. Example: Registration.Grade $\rightarrow$ Registration.Satisfaction.

2. Type II edges For any attribute $X.A$ and any of its parents $X.\mathbf{K}.B$, an edge from $Y.B$ to $X.A$ is introduced, where Y = Range[X.**K**]. Example: Course.Difficulty $\rightarrow$ Registration.Grade.

It is then required that the CDG is acyclic. When this requirement is met, the following statement holds:

If the CDG G_π is acyclic for a PRM Π, then for any skeleton σ_r the instance dependency graph is acyclic.

When this is met, a PRM can be called *coherent*.

Of course, there are exceptions to acyclicity in the CDG. [GFK+07] take the example of genetics: a person genetics depend on the genetics of the person's parents. A PRM of this domain is found in the left graph of figure 3.17.

So why does this cyclicity in a PRM work? Well, it depends on the constraints active on the domain itself. A person genetics could never be depended on his own genetics, but always on his parents'. Because a person's parents are a different instantiations of the schema, these

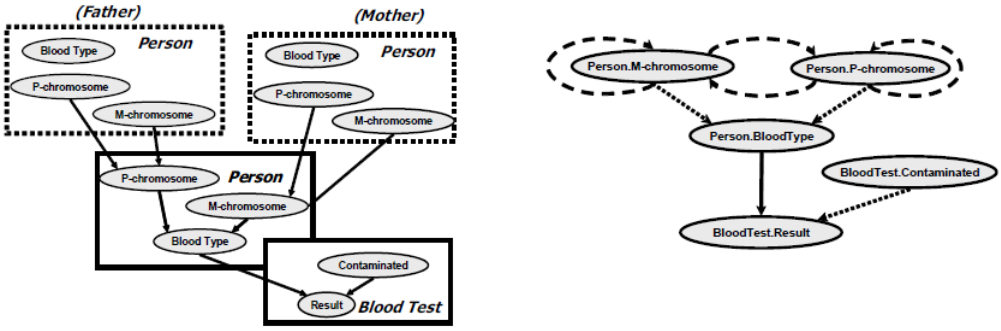


Figure 3.17: PRM for the genetics domain (left) and dependency graph with the three different colored dependencies (right) [GFK+07].

dependencies do not create a troublesome cycle. To create a more general theory on these dependencies, a 'color' theory is created by [GFK+07]. An example can be found in the right graph of figure 3.17.

Definition 20. To implement this theory, we first need the notion of *guaranteed acyclicity*. This means: certain reference slots $\mathcal{R}_{ga} = \{\rho_1, ..., \rho_n\}$ have such a partial ordering $\prec_{ga}$ that if y is a ρ-relative for some $\rho \in \mathcal{R}_{ga}$ of x, then $y \prec_{ga} x$ (the relation $y \to x$ is guaranteed acyclic). Guaranteed acyclicity of a slot chain $\mathbf{K}$ can then be achieved if each component ρ is guaranteed acyclic.

The colored class dependency graph can now be implemented:

Definition 21. G_Π is a *colored* class dependency graph for a PRM Π with the following edges:

1. Yellow Edges If $X.B$ is a parent of $X.A$, it is a yellow edge $X.B \to X.A$ (dotted arrows in 3.17.right).

2. Green Edges If $\gamma(X.\mathbf{K}.B)$ is a parent of $X.A$, Y = Range[$X.\mathbf{K}$] and $\mathbf{K}$ is guaranteed acyclic, it is a green edge $Y.B \to X.A$. γ is the aggregate function over the multiset (dashed arrows in 3.17.right).

3. Red Edges If $\gamma(X.\mathbf{K}.B)$ is a parent of $X.A$, Y = Range[$X.\mathbf{K}$] and $\mathbf{K}$ is not guaranteed acyclic, it is a red edge $Y.B \to X.A$ (solid arrows in 3.17.right).

There can be several edges of different colors between attributes. Therefor, attention needs to be paid at which edges are between attributes. If it are only yellow and green edges, acyclicity can be guaranteed. When red edges are in play, cyclicity can be the case and more knowledge about constraints on the domain need to be gathered to see if the probability calculation still works.

An other way to guarantee acyclicity is by using the *stratification* theory. If there is a stratification (existence of layers) between attributes of different classes and parents of an attribute proceed the attribute in the stratification ordening, acyclicity can be guaranteed.

A general definition for this is given by [GFK+07] as follows:

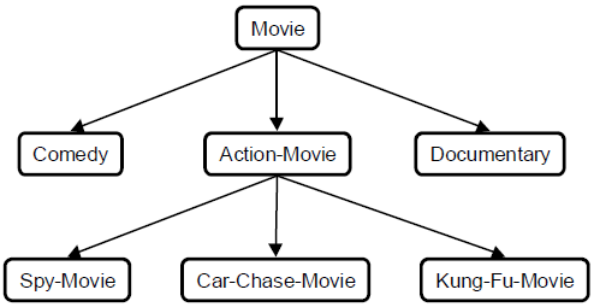


Figure 3.18: Class hierarchy of movie domain. The arrows depict *is-a* relationships [GFK$^+$07].

Definition 22. A (colored) dependency graph is stratified if every cycle in the graph contains at least one green edge and no red edges.

From this we can conclude: If the colored class dependency graph is stratified for a PRM Π, then for any skeleton σ_r, the instance dependency graph is acyclic.

3.4.4 Implementation of Class Hierarchy in the Framework

As with [KP97] and [BW00a], [GFK$^+$07] discuss the usage of *class inheritance* in their framework. They call it PRMs-CH (Probabilistic Relational Models with Class Hierarchies).

The reason for using CH is the same as it is for [KP97] and [BW00a]: making it easier to define classes which differ in only some attributes from other classes. Also, they claim by using CH, relations that normally couldnt't be expressed without CH, can be expressed with CH. This also means that for using CH, redefinition of existing formulas is needed and new concepts need to introduced.

As we could see in figure 3.5, CH depend on the Isa-relation. [GFK$^+$07] make use of an other example, namely the movie-domain. The Isa-relations for this domain are shown in figure 3.18. Formally, we can state the following about CH:

Definition 23. If we have a class X and a hierarchy H, $H[X]$ is a rooted directed acyclic graph, defined by a subclass relation $\prec$ over a finite set of subclasses $\mathcal{C}[X]$. X is the root of the class hierarchy. For $c, d \in \mathcal{C}[X]$, we say X_c is a *direct subclass* of X_d if $c \prec d$ (and X_d is a *direct superclass* of X_c). $\prec^*$ defines an *indirect subclass* relation. If $c \prec^* d$, X_c is an (*indirect*) *subclass* of X_d.

An example from the movie-domain: a Comedy is a direct subclass of a Movie. A Spy Movie is an indirect subclass of Movie.

To define all the subclasses that belong to a certain class we define a subclass indicator $X.Class$, indicating all objects $x \in X$ for which holds $x.Class \prec^* c$. In other words: all objects that are in a class which is a subclass of c. From the movie-domain, $Movie.Class$ would give the following set (see figure 3.18): $\{\mathsf{Comedy}, \mathsf{Action\ Movie}, \mathsf{Documentary}, \mathsf{Spy\ Movie}, \mathsf{Car-Chase\ Movie}, \mathsf{Kung-Fu\ Movie}\}$.

With the subclasses we are able to make more specific distinctions when building a probabilistic model. It enables the specialization of CPDs for different subclasses in the hierarchy. A PRM-CH can now be defined [GFK$^+$07]:

Definition 24. For each class $X \in \mathcal{X}$ of a PRM-CH we have

- a class hierarchy $H[X] = (\mathcal{C}[\mathcal{X}], \prec)$
- a subclass indicator attribute $X.Class$
- a CPD for $X.Class$
- for each subclass $c \in \mathcal{C}[X]$ and attribute $A \in \mathcal{A}(X)$ we have either
  - a set of parents $\mathrm{Pa}^c(X.A)$ and a CPD that describes $P(X.A|\mathrm{Pa}^c(X.A))$; or
  - an inherited indicator that specifies that the CPD for $X.A$ in c is inherited from its direct superclass. The root of the hierarchy cannot have the inherited indicator. This is because the root is already the top of the hierarchy and has nothing to inherit.

For each subclass, we can now specialize the CPD for its attributes. Also, for two subclasses, the parents of an attribute can be completely different. The popularity of an action movie can depend on its budget, while the popularity of a documentary can depend on its director.

Refining the Slot References

There is another advantage coming from the subclass representation. We are now also able to refer to a certain subset of an attribute. For example, if people could vote for movies, could we find correlations between them? By using subclasses we can specifically find votes for certain kinds of movies, which without subclasses would be impossible. If we want to find a correlation between a person's vote for action movies and for documentaries, we can now do so.

We can create subclasses of *Vote* for the different kind of movies we have. We than get a $\mathsf{Comedy} - \mathsf{Vote}$ (for comedies), an $\mathsf{Action} - \mathsf{Vote}$ (for action movies) and a $\mathsf{Documentary} - \mathsf{Vote}$ (for, you guessed it, documentaries). With this, we can isolate a person's vote for certain genre of movies and introduce a dependency between $\mathsf{Documentary} - \mathsf{Vote.Rank}$ and $\mathsf{Action} - \mathsf{Vote.Rank}$ (rank is the ranking of person gave to the movie).

But to do so, a mechanism is needed to restrict the types of objects that travel through reference slots to belong to specific subclasses. So we need to refine the slot-chains as follows:

Definition 25. If we have a slot ρ of X with range Y and subclass d of Y. The refined slot reference ρ_d for ρ to d is the following relation between X and Y: For $x \in X$, $y \in Y$, $y \in x.\rho_d$ if $x \in X$ and $y \in Y_d$, then $y \in x.\rho$.

We can now use the refined slot chains to fully exploit class hierarchies.

How to implement Instance-Level Dependencies

By using subclasses, the class-based probabilistic model of a PRM can now come closer to the instance-based Bayesian Networks. As example: the subclass hierarchy of movies starts very general and can go deeper and deeper, to the point you are at instance level. So we can define

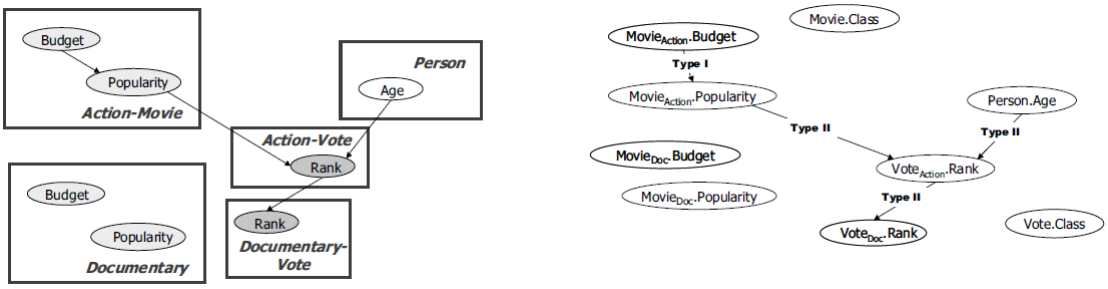


Figure 3.19: PRM with class hierarchy for the movie domain (left) and class dependency graph for the PRM (right) [GFK+07].

class based dependencies (a person enjoying action movies is likely to enjoy documentaries) or instance based dependencies (someone enjoying Star Trek is likely to also enjoy Star Wars).

And, with the redefines slot chains, we cane even combine class based and instance based dependencies. For example, if someone likes Star Wars, he is very likely to like to whole genre of SciFi-movies.

We can now give a redefinition of definition 18, including subclasses:

Definition 26. $P(\mathcal{I}|\sigma_r,\ \Pi)\ =\ \prod_{X}\ \prod_{x\in\sigma_r(X)} P(x.Class)\ \prod_{A\in\mathcal{A}()X} P(x.A|\mathrm{Pa}^{x.c}(x.A))$.

As before, we will not explain the formula. It is just there to show the probabilistic theory behind it.

Coherency of the PRM with Class Hierarchy

As was done with the PRM without CH, the PRM-CH needs to be coherent: it has to be checked for cycles. Checking only the relational skeleton is not enough, as the subclass indicator is not specified in it. This makes checking the PRM for coherence a lot more difficult than it was without CH.

So a new edge theory is added to the existing theory, with a corresponding *colored instance dependency graph*:

Definition 27. A *colored instance dependency graph* for a CH-PRM Π_{CH} and its relation skeleton σ_r is a graph G_{σ_r} with the following nodes for each class X and for each $x\ \in\ \sigma_r(X)$:

1. A descriptive attribute node $x.A$ for every descriptive attribute $X.A\ \in\ \mathcal{A}(X)$.
2. A subclass indicator node $x.Class$.

The dependency graph then contains four edges: two for the descriptive attributes and two for the subclass indicator. These two types of edges are stated below [GFK+07]. An example of the usage of the edges can be found in figure 3.19:

- **Type I edges** For every $x\ \in\ \sigma_r(X)$ and for each formal parent $X.B\ \in\ \mathrm{Pa}^*(X.A)$, we define an edge $x.B\ \rightarrow\ x.A$. This edge is *black* if the parents have not been specialized. This is the the case for the subclass indicator $x.Class$ and possibly other attributes. All other edges get the color *gray*.

- **Type II edges** For every $x \in \sigma_r(X)$ and for each formal parent $X.\mathbf{K}.B \in \text{Pa}^*(X.A)$ (**K** is a slot chain), if $y \in x.\mathbf{K}$ is in σ_r, there is an edge $y.B \rightarrow x.A$. If the CPD has been specialized or **K** contains any refined slot references, the edge is colored *gray*. If not, the edge is colored *black*.

So, we now have two types of edges. In practice, they say the following thing: the black edges are known to exist. Gray edges, however, indicate that we do not know if the edge exist in the CPD. With only black edges, it is easy to depict whether the graph is acyclic or not. The gray edges make it much more problematic.

So the notion for acyclicity of the colored instance dependency graph needs to be extended:

Definition 28. A colored instance dependency graph is acyclic if, for any instantiation of the subclass indicators, the ordering of the nodes connected to black edges in the graph is acyclic. A black edge is then determined in the following way:

- With a subclass assignment $y.Class$, all the edges involving this object can be black or white. With $y.Class = d$, the edges for any parent nodes are colored black if the CPD $\text{Pa}^d(X.A)$ has defined them. If not, the edges are colored white. The edges belonging to any refined slot references, $\rho_d(x, y)$, are colored black if $y.Class = d$. If not, they also become white.

If a PRM with class hierarchy now has a acyclic colored dependency structure over his relational skeleton, we can state that the PRM and its relational skeleton define a coherent probability distribution over the instantiations that extend the relational skeleton.

In section 3.4.3 we showed that we can ensure that any relational skeleton over a PRM is acyclic. Therefor, we need to extend the definition of the class dependency graph:

Definition 29. The class dependency graph for a PRM with a class hierarchy Π_{CH} has the following set of nodes for each $X \in \mathcal{X}$

- For each subclass $c \in \mathcal{C}[X]$ and attribute $A \in \mathcal{A}(X)$ there is a node $X_c.A$.
- A node for the subclass indicator $X.Class$.

and the following edges:

- **Type I edges** For any node $X_c.A$ and formal parent $X_c.B \in \text{Pa}^c(X_c.A)$ there exists an edge $X_c.B \rightarrow X_c.A$.
- **Type II edges** For any attribute $X_c.A$ and formal parent $X_c.\rho.B \in \text{Pa}^c(X_c.A)$, where Range[$\rho$] = Y, there exists an edge $Y.B \rightarrow X_c.A$.
- **Type III edges** For any attribute $X_c.A$ and for any direct superclass d, $c \prec d$, we add an edge $X_c.A \rightarrow X_d.A$

With these extension we can now define a PRM with Class Hierarchy where the class dependency graph of the PRM is acyclic for any relational skeleton.

3.4.5 Findings on the Framework

In comparison to [KP97] and [BW00a], [GFK+07] give a very database and class based implementation of PRMs. Their initial schema isn't related to a BN, but to what we now see as an UML Class Diagram. As with [BW00a] a class contains attributes (nodes in the framework of [BW00a], which can be classes themselves.

But in [GFK+07]'s framework there isn't a layered structure with input-, output- and internal nodes. Instead, attributes are directly linked to the attributes they influence. Still, reference slots (which were also nodes in [BW00a]) are needed to create these links.

On of the most important findings on [GFK+07]'s framework: it is very graphical. Where [KP97] lacks in graphical representation and [BW00a] still haven't optimized their graphical part, the framework from [GFK+07] is almost completely graphical, backed by a clear and lengthy explanation of logic and probabilistic calculations behind it. The only shortcoming in graphical representation was with the class hierarchy. There was no example about how a class hierarchy could be represented in their graphical framework itself. The best they could do was a separate model.

Most noticeable about the class hierarchy was that the theory behind the model becomes a lot more complex. We tried to show the workings of these formal theories, but did not try to explain them in length. This is because this thesis is about the graphical representation of things and a lot less about the formal representation of it on paper.

Still, we see the same problems the other two frameworks had: they still lack a good base to put constraints on their framework and not all of the graphical representation was present.

3.5 Concluding

Now we discussed both frameworks, some general remarks can be made. First of all, the first two frameworks enable you to use Object Orientation in combination with Bayesian Networks. Where [KP97]'s framework stays very close to its mathematical base, [BW00a] focus less on the calculations and more on the manner to representations objects in their framework.

This allows [BW00a] to be less complicated than [KP97]'s, making it more usable and understandable. Still, both frameworks lack in representing varying objects and constraints on certain relations or objects.

The framework by [GFK+07] shows a much more modern approach, using a UML CD based view on relational models. They then implement the PRM in this RM. Because of the easy nature of their framework and the relation to CD, the framework of [GFK+07] will be the one we will use to see if ORM can be combined with PRM. This is one of the reasons why we discussed this framework in more length and with more mathematical background. The other is that [GFK+07] discussed their framework in much more detail.

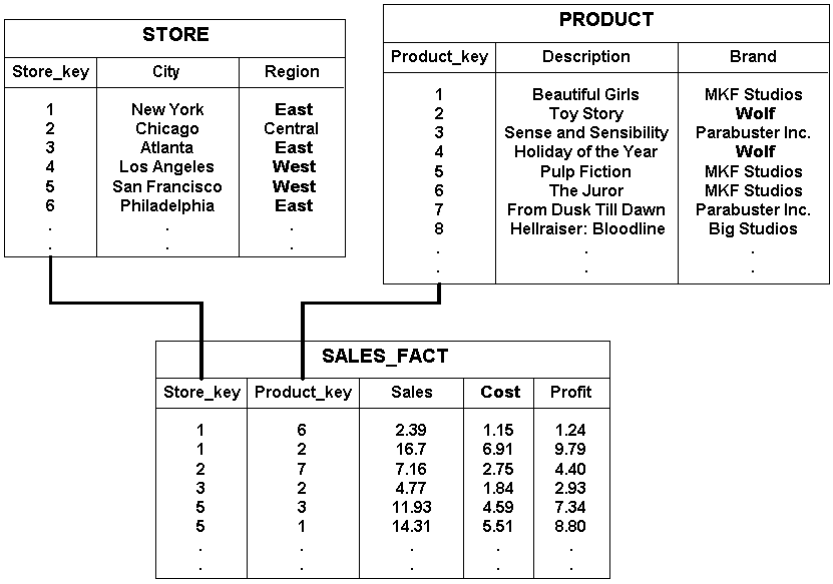
Now we've discussed the history of PRMs, their definitions and specified what they look like, we now continue to specify the modeling language used to possibly extend PRMs: ORM.

[KP97]	[BW00a]
General Definitions	
Class = {attributes} Attribute = Single Object | Object Single Object = Random Variable	Class = {nodes} Node = Simple Node | Instantiation Simple Node = Random Variable
Explanation: The general definitions are the same, but differ in terminology. By using *Object* or *Instantiation* multi-layered objects/nodes can be constructed.	
Typing	
Basic Type = value Structured Type = Basic | Structured Type	Type = value
Explanation: While [KP97] make use of structured types to construct multi-layered objects, [BW00a] construct multi-layered objects from Instantiations and Simple Nodes.	
Inner Specification	
$Object = \begin{cases} Inputs \mathcal{I}(X) \\ Values \mathcal{A}(X) \begin{cases} \text{Encapsulated } \mathcal{E}(X) \\ \text{Outputs } \mathcal{O}(X) \end{cases} \end{cases}$	$Class = \begin{cases} \text{Input Nodes } I_X \\ \text{Internal Nodes } H_X \\ \text{Output Nodes } O_X \end{cases}$
$\mathcal{I}(X)$ are Single Objects $\mathcal{E}(X)$ are objects $\mathcal{O}(X)$ are objects	I_X are reference nodes H_X are real nodes or instantiations O_X are real or reference nodes
Each attribute has a structured type.	$\begin{cases} I_X \text{ are simple nodes} \\ H_X \text{ are simple nodes or instantiations} \\ O_X \text{ are simple nodes} \end{cases}$
Explanation: In [KP97]'s framework, input and output objects are structured types. For [BW00a] to have the same expressiveness, they should allow input and output nodes to be instantiations. Instead, they make use of reference links, which, with some adaptations, can express the same thing.	
OOBN Definition	
OOBN = Objects with a single situation object	OOBN = a class
Explanation: Again, the situation object of [KP97] pops up. As said in the section discussing their framework, this situation object is not well defined, but can be best seen as an structured object. As can be seen from the definition, an OOBN from the eyes of [BW00a] is simpler than an OOBN from [KP97]	

Table 3.1: Differences and Similarities between [BW00a] and [KP97]. Table based on [BW00a]

Chapter 4

What is Object Role Modeling (ORM)?



STORE

Store_key	City	Region
1	New York	**East**
2	Chicago	Central
3	Atlanta	**East**
4	Los Angeles	**West**
5	San Francisco	**West**
6	Philadelphia	**East**

PRODUCT

Product_key	Description	Brand
1	Beautiful Girls	MKF Studios
2	Toy Story	**Wolf**
3	Sense and Sensibility	Parabuster Inc.
4	Holiday of the Year	**Wolf**
5	Pulp Fiction	MKF Studios
6	The Juror	MKF Studios
7	From Dusk Till Dawn	Parabuster Inc.
8	Hellraiser: Bloodline	Big Studios

SALES_FACT

Store_key	Product_key	Sales	Cost	Profit
1	6	2.39	1.15	1.24
1	2	16.7	6.91	9.79
2	7	7.16	2.75	4.40
3	2	4.77	1.84	2.93
5	3	11.93	4.59	7.34
5	1	14.31	5.51	8.80

Figure 4.1: An example of some tables in a database

ORM finds its origins in the 70's, evolving from the *Natural language Information Analysis Method* (NIAM) developed by Dutch researcher G.M. Nijssen and also evolving out of research from other scientists. It was Terry Halpin who first spoke of ORM in 1989, writing his PhD-thesis. We will discuss ORM based on Halpin's article from 1998: *Object Role Modeling (ORM/NIAM)* [Hal98]. There are a lot of additions for ORM, expanding it on almost every aspect. For this thesis we will only use the basis of ORM as described by [Hal98], this gives us enough detail about ORM to put it to work regarding PRMs. First, it is discussed what ORM exactly is, with [Hal98] as main source. The second section describes the workings of ORM and its graphical representation. The third section will give an example of building an ORM schema and the last section will specify why we use ORM for this thesis.

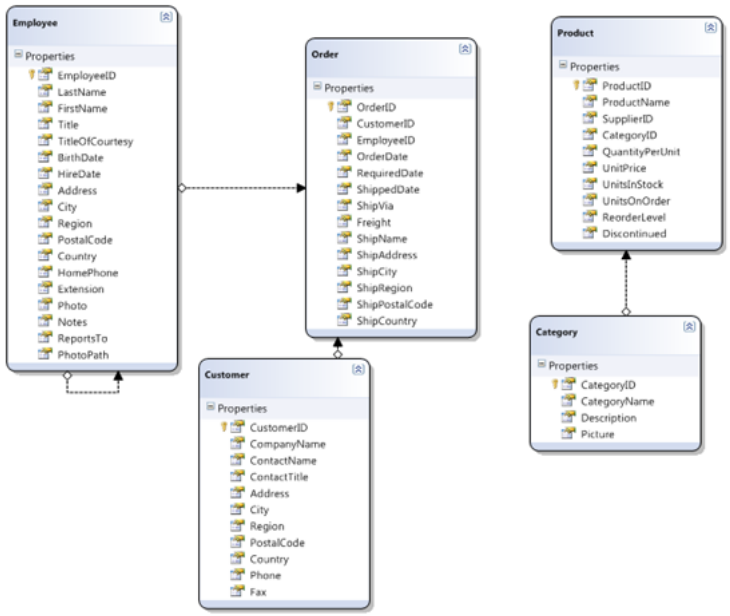


Figure 4.2: An example of a logical model behind a database.

4.1 What is ORM?

ORM was created as a method to model an information system at the conceptual level. As seen in the introduction of this section, it is also known as NIAM (mostly in Europe). ORM's main point of view is databases with some logical data model behind it. Databases can be seen as tables (figure 4.1) filled with data, structured by a logical data model (figure 4.2).

The strength of ORM lies in its ability to be defined at conceptual level, making use of a notation and a language that is very natural to use for most people. It uses natural language as well as easy to use diagrams to model the Universe of Discourse (UoD): the application area of the model.

The name ORM defines the models it produces: it makes use of *objects* (lets say a car and a person) which are in some kind of *relation* (like Person has Car). A *role* is a part in a relationship. In the small example: the person plays the role drives in, the car plays the role gets driven by. The two roles together make the relationship (or *fact*) driving. Interesting fact: the relationship itself is also an object, which can also play a role in other facts.

ORM differs from other modeling techniques, like Entity-Relationship(ER) and Object Oriented languages (OO-languages), because of the absence of attributes in ORM. Attributes are the same level *object types* as the class the attribute is in. In our small example: instead of using hasCar as attribute of Person, ORM creats a fact Person drives in Car. According to [Hal98] the three most important advantages of this representation are:

1. ORM models are more stable: attributes may evolve into *entities* or relationships, which would mean in, for example ER, that attributes have to be redefined. In ORM, attributes are already relationships or entities.

2. ORM models may be conveniently populated with multiple instances.

3. ORM is more uniform (e.g. a separate notation for applying the same constraint to an attribute rather than a relationship is not needed)

A big disadvantage of using OO-models is that they have a weak support for constraints. This also backed by [KP97], giving the limitations of their framework. OO-models are also less stable when it comes to a evolving UoD (see point 1 from the enumeration above).

So, ORM is a language, using *objects* and *roles*, defining entities and relationships between them. This has as big advantage over ER- and OO-models: ORMs can adapt well to changing UoDs (read: domains), which makes them more stable.

4.2 Graphical Representation of ORM schemas

In this section the graphical representation of ORM will be discussed. Not all symbols discussed from figure 4.3 are used in this thesis, but to fully understand the possibilities of ORM, all symbols in figure 4.3 are explained.

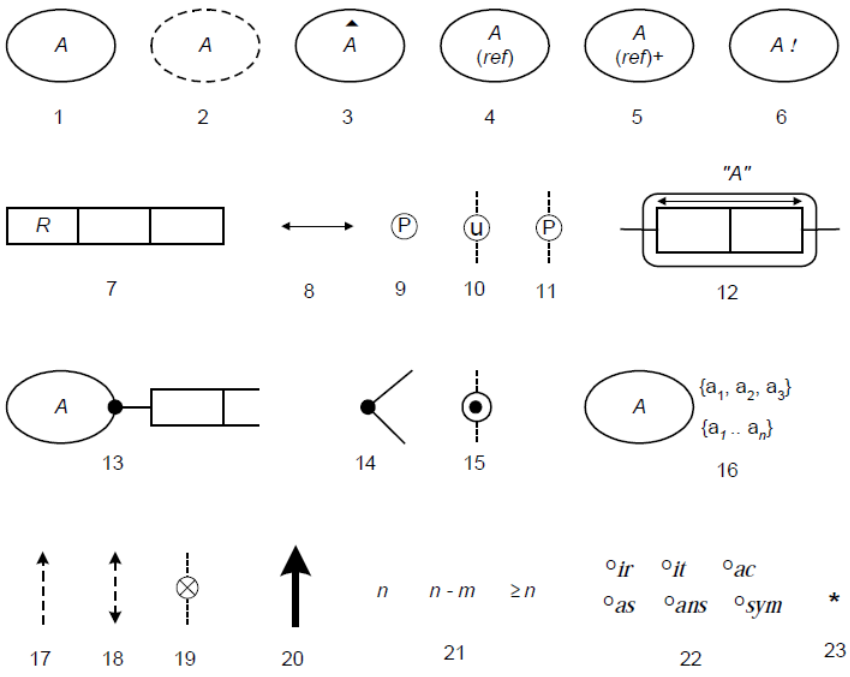


Figure 4.3: Main ORM Symbols [Hal98]

1. **Entity Type** (ET) named A.

2. **Value Type** (VT) is a entity type with a specific value, like a string or integer. Values are commonly known, like integer-based values: time, years, costs in money etc. Also

values like booleans and names can be seen as value types, being known to many. The VT therefor do not require a reference scheme.

3. **Arrow Tipped ET** The object appears more than once in the schema.

4. **ET with Reference** Each object needs to have a reference scheme behind it, which indicates how it can be mapped to one or more values behind it. When an object has only one value type behind like (like an ID or a Name), the reference schema can be displayed in the entity type itself.

5. **ET with Reference+** The plus indicates that the value is a numeric value.

6. **ET with exclamation mark** Indicates that an ET is independent, meaning it does not play a role in any facts. Normally, these ET are not introduced into schemes because they are not likely to be important.

7. **Ternary Predicate** (TP) Predicates are also known as facts. A TP contains three roles (displayed as a box), each played by exactly one object type. Roles are connected to their object type by a line segment (see symbol 13). A *predicate* can be seen as a sentence with holes where the object-values can be put in. For example ... is ...with ... can become (with the right object types connected to it): Brand is a VehicleType with Color. When filling in values, you could get: Brand *Honda* is VehicleType *Car* with Color *Blue*. The number of roles is called the 'arity' of the predicate (here it is ternary, with 2 roles it would be binary). If we treat a predicate formally, the order in which the roles are aligned matters: Color is Brand with VehicleType is something different than Brand is VehicleType with Color. When treating it as natural language, you see the first one is rubbish and use the second one.

8. **Internal Uniqueness Constraint** (IUC) are placed over one or more roles in a predicate and indicate that the (combination of) instances from the constrained roles are unique. For example, putting an IUC on Car in Car has Brand means that Car can only have one Brand (but a Brand can have more than one Car connected to it).

9. **Primary IUC** a predicate can have more than one UC, but only one is primary, which has the symbol *P*.

10. **External Uniqueness Constraint** (EUC) A uniqueness constraint connected to two or more roles from different predicates. A dotted line is used to connect them. By adding the constraint, the of combination of those roles in the join of the fact types becomes unique. For example, when we want to say that a state is identified by its statecode and country, we add an EUC on the roles played by Statecode and Country in State is in Country and State has Statecode.

11. **Primary EUC** To show an EUC is primary, you use a *P* instead of *u*.

12. **Objectification** If we want to treat a predicate as an entity type (and let it play a role in some kind of relation) we can objectify it by drawing a rectangle around it. They are also called *nested* object types.

13. **Mandatory Role Constraint** (MRC) This constraint is added to a role if all instances of the role's object type must play that role. The symbol is a black dot, placed on the connection between the ellipse and the relation-line. It is also called a *total role constraint*.

14. **Disjunctive MRC on ET** This Disjunctive MRC has the same meaning as (15), but is attached to the roles on the Enity Type itself. This representation is not commonly used as it is not that clear.

15. **Disjunctive MRC** This MRC can be connected to two or more roles (from the same object type) of different predicates. It means that all instances of this object type must play a part in at least one of those roles.

16. **Value Constraints** This constraint is used to limit an object type's population to a list of values. This is done by putting the values in braces next to the object type (symbol 16, upper). If the list is ordered, you can show the first and last value of the list and put '...' between them (symbol 16, bottom).

17. **Subset Constraint** (SC) may only be applied between two compatible roles (these are roles with the same object type as host). The arrow goes from one role to the other, telling that the the population of the first role is a subset of the other. So, if we have role A and role B, A –> B means $A \subseteq B$.

18. **Equality Constraint** (EC) may only be applied between two compatible roles. Means the population of the two roles is equal.

19. **Exclusion Constraint** (ExC) may only be applied between compatible roles. Means that if an instance plays a part in one role, it may not play a part in one of the other roles connected with the ExC. In other words: the ExC displays mutual exclusiveness.

20. **Subtype** the solid arrow is drawn from one object type to the other, displaying the first is a subtype of the other.

21. **Frequency Constraint** Applied to a sequence of one or more roles, these indicate that instances that play those roles must do so exactly n times, between n and m times, or at least n times.

22. **Ring Constraint** can be added to a pair of roles played by the same host type. Indicates that the binary relation formed by the role population must be irreflexive (*ir*), intransitive (*it*), acyclic (*ac*), asymmetric (*as*), antisymmetric (*ans*) or symmetric (*sym*). Is not commonly used in ORM.

23. **Derivable Fact Type** can be placed beside a fact type, showing that this fact type is derivable from other fact types. Is also not commonly used.

To get a better understanding in how to build a model with this graphical representation, the next section will model our RE in ORM, starting from the starting point of all models: the domain.

Step

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.

Figure 4.4: The conceptual schema design procedure by [Hal98]

4.3 Building our RE as an ORM schema

Remember our RE, defined in section 3.1? Let's start from scratch and build this model as an ORM model. We use the steps mentioned in figure 4.4 found in [Hal98].

'Step 0' First of all, we add a 'step 0' to this procedure: defining the domain. The domain of the RE is sports and more specific sporting teams, like football clubs, (ice)hockey teams and even cricket teams. To restrict the domain, we will only look at one sporting team. Possible extensions of the domain could be a league of sporting teams or all sporting teams of a sport in one country. But to keep the model small, we will limit ourselves to one sporting team. Besides that, we limit the connections (relations) in the domain a bit. This is because, as you will see, a small scope like this can already give a quite large ORM-schema.

Step 1 As defined in section 3.1 a Sporting Team consists of Players and a Trainer. Furthermore, a Sporting Team is based in a Country (when thinking about national teams, the sporting team can be a country itself). We can now define the following elementary facts:

1. Sporting Team has Players
2. Sporting Team has Trainer
3. Sporting Team is based in Country

Of course, this is not all information we know. A Sporting Team also performs in some kind of way, has a Fanbase, makes a Revenue and wins Prizes. So, we know the following four new facts:

4. Sporting Team has Performance
5. Sporting Team has Fanbase
6. Sporting Team makes Revenue
7. Sporting Team has won Prize

According to the definition of our RE, we now have included all the important objects in our description. But we are not done yet. In ORM every entity type needs a reference scheme, which tells us which values it contains and what identifies the entity type. Let's start with Players.

Players are a set of one or more Player-instances:

8. Players contains one or more than one Player

But what is a player? Well, as mentioned before when treating the RE, a player can be seen as a person with some extra attributes:

9. Player is a Person

So which attributes does Person have and which extra attributes does Player have? Person has a name, an age, a height and a weight. A normal Person is also likely to have an address, hair color and an eye color. But these are not really important for the sports domain (for most sports, performance is more important than looks). Player can have the extra attributes speed, strength and talent. If you would go deeper, you could specify different attributes for all different kind of sports. You can than create a subtype of Player for every type of sport. To keep this example small, this RE will no be that detailed. We now know these new facts:

10. Person has Name
11. Person has Age
12. Person has Height
13. Person has Weight
14. Player has Speed
15. Player has Strength
16. Player has Talent

The same goes for Trainer, which is also a Person. And has some extra attributes, like experience, trainer skill and prizes he won as trainer/player. This gives us new facts:

17. Trainer is a Person
18. Trainer has Experience
19. Trainer has TrainerSkill
20. Trainer has won Prizes

Also related to Person is the Fanbase. A Fanbase consists of Fans and they are also a subtype of Person, with some extra attributes. A fan has a favorite sporting team and is supporting that team for some amount of time (in most cases for some years). A fan could also have a season ticket, which allows him to watch games regularly. This gives the following new facts:

21. Fanbase consists of one or more than one Fan
22. Fan is a Person
23. Fan supports Sporting Team for Time
24. Fan owns Season Ticket of Sporting Team

Besides the connections Sporting Team already has, it is also good to know which sport is played by the sporting team. Prizes is a collection of one or more than one Prize with a name, won in a certain year in a certain sports-category by a certain team:

25. Sporting Team plays Sport
26. Prizes is one or more than one Prize
27. Prize has Name
28. Sporting Team playing Sport won Prize in Year

The last thing to do is to look at how each entity type is identified (they all need a reference scheme of values they can contain). In case of Person, Player, Trainer and Fan: they are all referenced to by their name. Because people can have the same name, it is not possible to uniquely identify a Person by only its name. But we will discuss this in step 3.

The Revenue is an integer, displaying the yearly revenue of a sporting team. Country is referenced to by the country's name. Prizes and the Sporting Team itself are also referenced to by their name.

Step 2 We now can create a first ORM-model from the facts found in step 1. To create the model, VisioModeler was used. The result is found in figure 4.5. As you can see the symbols from figure 4.3 are used.

Almost all facts in the model are binary (a fact type with two roles), with two entity types connected to it (for example: Sporting Team has Players). There are also some value types, for example the objects Revenue and Year. Three ternary fact types are in the model, the representations of fact 22, 23 and 27 in step 1. The most interesting of the three is Prize. This is fact type is an *objectification*, describing that a Prize consists of a Prize Name, the Sport it was won in and the Year in which it has been won. There is also an UC on this fact type. This UC should not yet have been added, this is done in the next step. But when objectifying a fact type in VisioModeler, the UC is added automatically.

Another interesting symbol in the model are the two power types Fanbase and Players. They contain a set of entities from the objects Fan and Player and represent facts 8 and 20 from step 1. A population check was not performed, because we are not building an actual database from this model and that is the reason to do a population check. Now we have created an ORM-model from our facts, we can go to step 3.

Step 3 This step does not apply for this model. There are no entity types we would want to combine. And there also no objects that could be derived from others. So, we skip this step.

Step 4 & 5 Constraints can now be added to the model. Because both step 4 and 5 are about applying constraints, we combine this to one step. The resulting model can be seen in figure 4.6. When building a database out of an ORM model, every fact type should have uniqueness constraint on it. These constraints help in defining what the identifying (combination of) attribute(s) is (the unique key). For example, in the RE, a person is uniquely defined by the combination of the name, height, weight and age. If we would remove one of these attributes out the key, it is very likely the key isn't unique anymore. For example: if we would use only name and age and we want to look for Jan Janssen who is 20 years old, it is very likely there is more than one Jan Janssen in the database. By adding height and weight we ensure we find only the Jan Janssen we are looking for. In most databases a unique code is used as unique key, which solves the identification problem.

But, because it is not the intention to create a database out of this model, all the constraints added are constraints which would apply in the physical world. Take the fact Person has Age: here, the role Person plays is unique. This means: a person can only have one age. Which, of course, is consistent with what we see in our daily lives. The same logic can be applied to all other UCs. On the fact Sporting Team is in Country the UC is on both roles. Because there are sporting teams with the same name, it is possible that two teams with the same name are in different countries. This also means a team name is not able to uniquely identify one sporting team.

There are also some mandatory role constraints. Again, we take the fact Person has Age. The role Person plays is mandatory: every person has an age, so all instances in the entity type Person must play that role. When combining the MRC and UC you can conclude a person must at least and at most one age. Again, this logic can be applied to all other MRC in figure 4.6.

Step 6 Step 6 adds the value constraints to the model. For example: Trainer Skill is a level given to the trainer which indicates how capable he/she is in training a team. A Trainer can be *low* skilled, *medium* skilled or *high* skilled. Season Ticket is another story. If a fan owns a season ticket of a team, the value becomes *yes*, otherwise it is *no*. So, the value constraint limits the values of Season Ticket to *Yes* or *No*. The value constraint can also be found in 4.6.

Step 7 This step can also be skipped. We have added all the constraints needed. The optimization part is done when creating a database out of it. This optimization would mean: making sure that no needless tables are created and tables which can be combined will be combined. Because this fall out of the scope for this thesis, we will not execute this step.

4.4 Why ORM is used in this Thesis

Modeling domains can be done in many different ways with many different languages. Besides ORM there are hundreds of other languages to be used. The most standard alternative for ORM is the Unified Modeling Language (UML) [Coo00] and in specific its Class Diagrams. UML is the most popular modeling language used and consists of many different models and diagrams, which are able to model all aspects of a domain, like work flow, activities and the

domain itself. In comparison to UML, ORM can mostly be only applied to modeling the domain itself.

The main reason to use ORM in this thesis is because it has not yet been used for expressing PRMs. [SEJ10] discusses the usage of UML as relational model for PRMs and succeeds pretty well in it. Because ORM is one of the most expressive modeling languages around, it seemed interesting to try the same with ORM and see what would happen.

What advantage could ORM give over existing relational models used in PRMs? Well, there are still limitations in the existing PRMs: they lack representation of constraints and are not able to express a variable amount of instances for their models. This problem comes forth from the fact that most PRM-frameworks make use of their own created domain modeling languages. These are all very basic, limiting themselves to objects, attributes and relations [KP97], classes, nodes and relations [BW00a] and classes and attributes [GFK+07]. It is, however, a shame they all neglect the strength of modern, Object Oriented modeling languages like ORM and UML. [SEJ10] does combine PRM with UML, showing that the combination of the two is not difficult. But even UML is very limited in expressing constraints, limiting the ability to express the complexity of domain. ORM is very able to do these tasks for complex domains. In chapter 5 we put to the test if these features of ORM can also be used in combination with PRMs.

One of the biggest advantages ORM has over other modeling languages is its ability in expressing relations. In ORM you can express almost every relation there can be expressed in a natural language. Besides being more expressive relation-language wise, it is also possible to objectify relations or create relations containing more than two roles. Even UML CDs do not give us the relation expressiveness ORM gives us. Besides being a big advantage, some caution is needed. There is always more than one right 'model' for a domain, especially in ORM. However, by giving the chance to create complex relations (relations with a lot of constraints and involved object types) there are also a lot of 'wrong' models and relations which you can create. The limitations of other languages in comparison with ORM and if the expressiveness of ORM can give a advantage over these other languages when using PRM with them, will be discussed by us in the next chapter.

4.5 Concluding

ORM is a modeling language used to represent domains by using objects and roles. The roles objects play are called facts, a fact being a semi-formal sentence like Person has Age. To graphically represent these facts, symbols are used to create a schema. In this schema, you can express the facts and add constraints to them. It is also possible to add a population to this schema and create databases out of the schema.

Because the lack in usage of modern Object Oriented domain modeling languages in general, the ability to express constraints specific and its expressiveness in relations, ORM is an interesting choice to use as graphical model to represent PRMs. The workings of this will be shown by us in the next chapter.

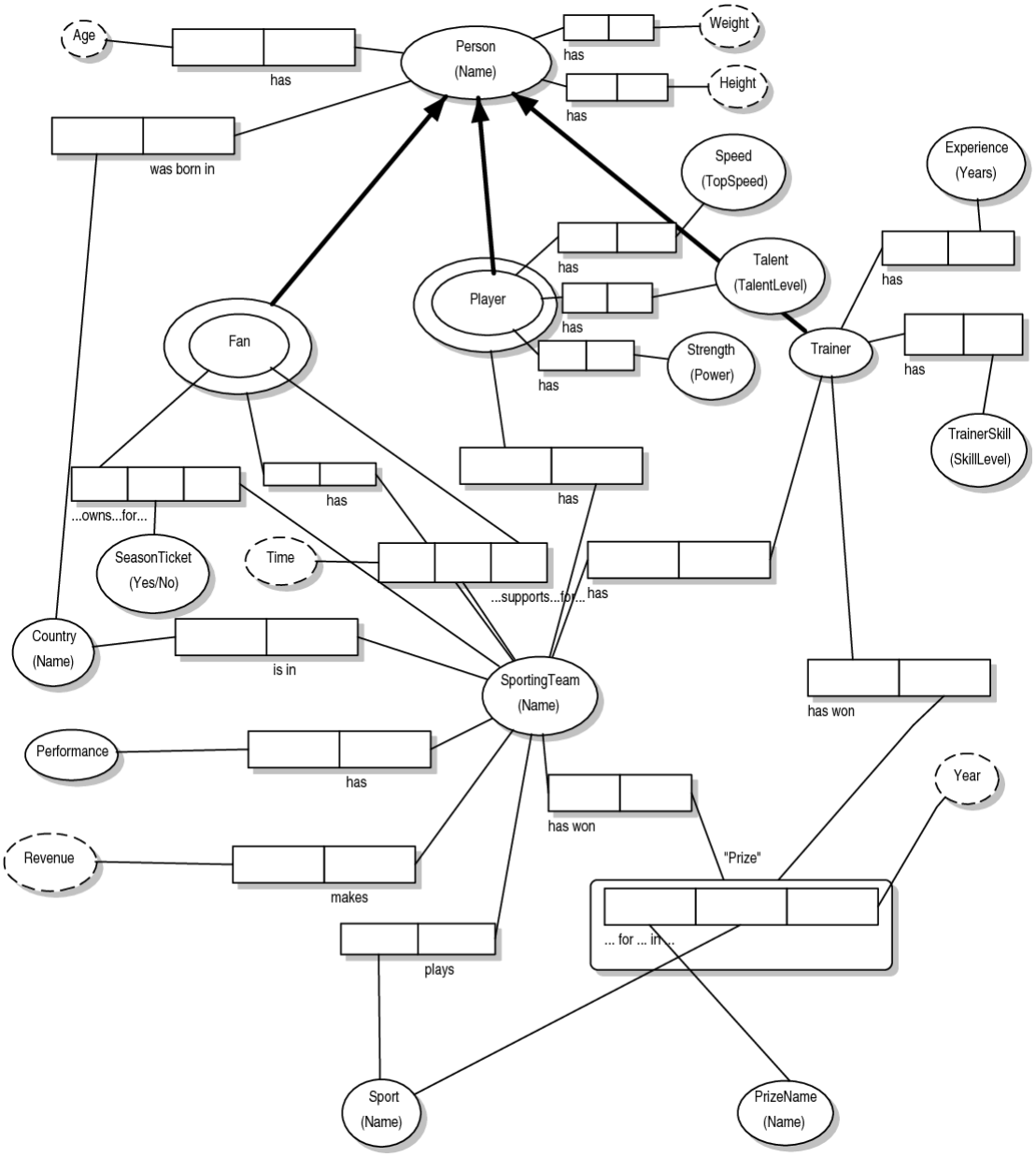


Figure 4.5: ORM model of Running Example with only facts

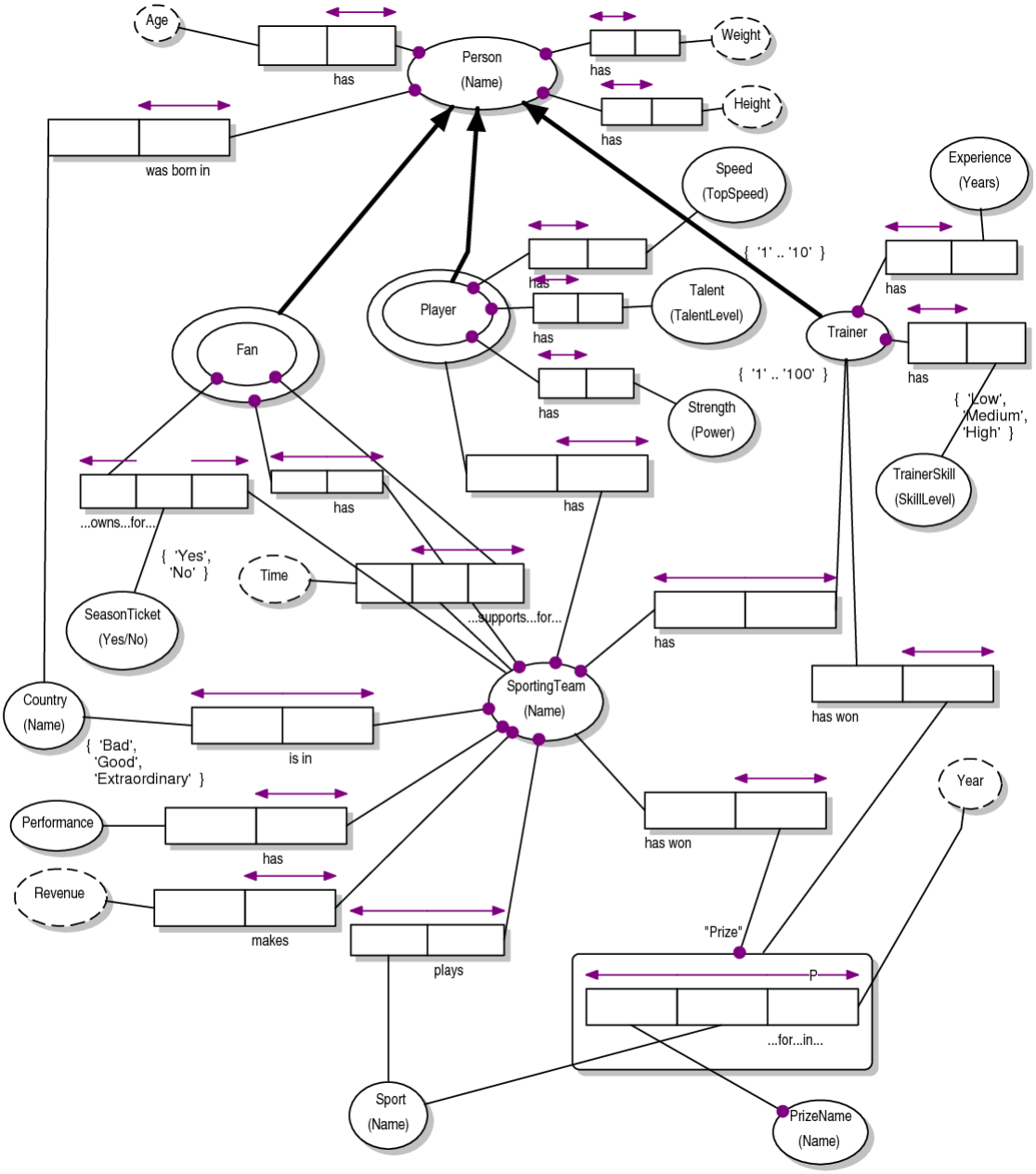


Figure 4.6: ORM model of Running Example with all constraints.

Chapter 5

ORM and PRM: a Working Combination with new Possibilities?

As we saw in chapter 2, the most important feature of a PRM model is that it contains two things: a relational model over a domain and a probabilistic component describing the probabilistic dependencies in that domain. This probabilistic component can also be seen as a Bayesian Network. The implementation of these two components of a PRM differs in every framework designed for it. This chapter will discuss how to represent a relational model and its probabilistic component in ORM.

We will first try to link the relational model from [GFK+07]'s framework (GFW) to ORM, building the base on which the PRM can be built. We then look at how the probabilistic model from GFW can be linked to ORM. The third section will look at possible advantages and disadvantages that ORM can have over the other modeling languages find in the literature. The fourth section concludes.

5.1 Converting the Relational Model: from GFW to ORM

This section will take a look at the similarities and differences in the relational model of the framework of [GFK+07] and ORM. Not all concepts in the first can be expressed in ORM. Still, we need to find some connections to see if a combination of ORM and PRM is possible at all. Constraints are not yet discussed in this section.

5.1.1 Converting Objects and Relations

When comparing [GFK+07]'s framework and ORM, *Classes* and *Attributes* in GFW are represented by *Objects* in ORM. If you look at figure 5.1, the class Person and its attributes Age, Weight and Height of (A) become entities in the ORM schema of (B).

A more difficult 'translation' is that of the relations of GFW. In GFW, the relational model has two kinds of relations: the class-attribute relation and the reference-relation. The first relation-type is not explicitly drawn with an edge or a link in GFW, but implicitly the link is there. For example, in figure 5.1.(A), if we would express the relation between the class Person and its attributes, in semi-natural language the relation would look like: Person has ..., with on the dots an attribute of the class. In ORM, this is translated with a binary relation

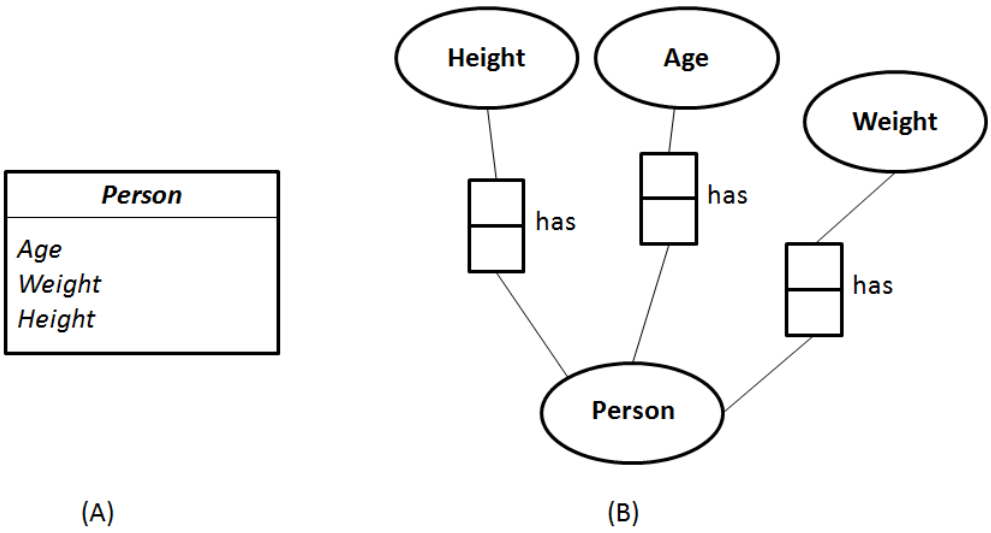


Figure 5.1: Objects and relations in GFW (A) and ORM (B).

(or fact) between two objects, shown in 5.1.(B). This translation is not very valid formally. In ORM, a relation can be read in both directions, meaning that Person has ... can also be read as ... has Person . In most cases, from the context of the model it can be concluded what the attribute is and what the class. Still, a formal definition is needed for a valid distinction between class and attribute. The translation of the implicit relations in GFW to explicit relation in ORM is what we call the ***Implicit-Relation Translation*** or ***IR Translation***.

The class-attribute relation from GFW can now be created as a has-relation in ORM. Then there are the reference slots in GFW. These slots enable the usage of other classes in a certain class. If we look at figure 5.2, (A) has two classes with reference slot Trainer in class Sporting Team. The translation of this class relation in ORM is found in (B). In GFW, the reference slots were given names, like SportingTeam.isTrainedBy.Trainer. isTrainedBy is then the name for the reference slot. Because ORM makes no use of reference slots, but directly links the class and the referenced attribute, we use the relation Sporting Team has Trainer. Of course, we could also write Sporting Team is trained by Trainer, but to explicitly show the class-attribute relation of it, we use has.

Reference slots in GFW were depicted as ρ. One of the possibilities was to invert the reference slot, ρ^{-1} (see definition 11). In ORM, almost every relation can be traveled in two or more directions (the more roles in a relation, the more directions a relation has). For example Sporting Team is trained by Trainer can also be seen as Trainer trains Sporting Team . Only unary relations, where only one role is involved, has only one direction.

Reference slots and slot chains, as defined by GFW, are not that important in ORM. This has to do with the absence of a class representation in ORM. Everything is an object type, classes and attributes. Therefor, a class is directly connected to attributes with a relation and if a class is used as an attribute in another class, the object type of the attribute and the class is the same.

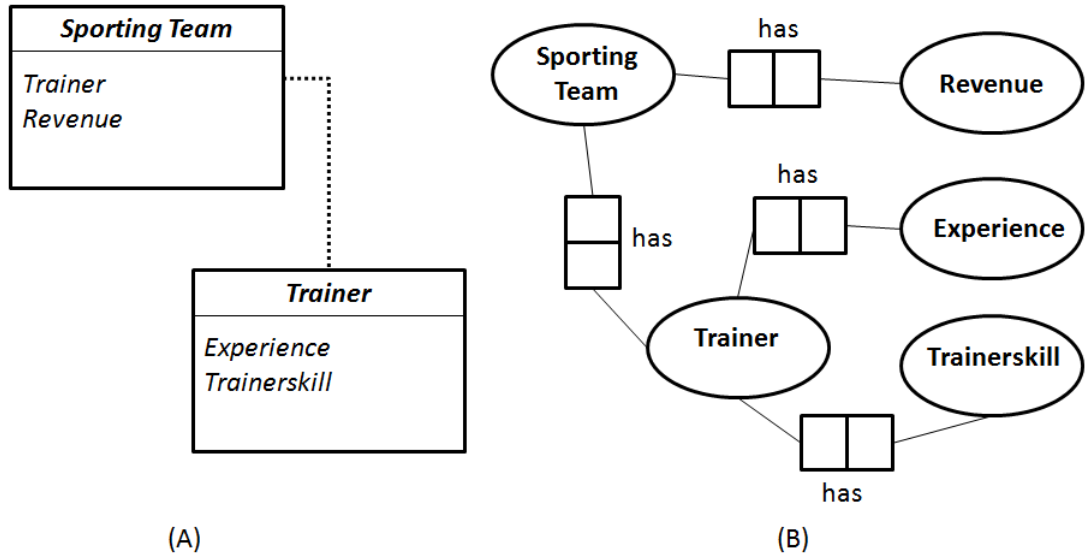


Figure 5.2: Two classes with reference slot in GFW (A) and the representation in ORM (B).

5.1.2 Converting the Classes

So, the relations GFW have in their relational model can be expressed in ORM. However, the class-based structure gives us more problems. In figure 5.1 and 5.2 we saw that ORM has no explicit graphical mention of classes. They are just object types like attributes are. As we named all the class-attribute relations has, we could specify a rule stating that all relations in the form of A has B, A is the class and B is the attribute. But because relations in ORM can be named everything you want (Sporting Team has Trainer could also be Sporting Team is trained by Trainer), this rule in most cases will not apply. Also, an ORM relation can be read both ways, making it harder to see what is the class and what is the attribute. The problems of ORM with class representations is what we will call the ***Class-Representation Problem*** or ***CR Problem***.

The question now arises if it is needed to make a distinction between classes and attributes in ORM. Because the later introduced probabilistic dependencies are not class bound, the class-attribute distinction is not that important. ORM works fine without classes. The only disadvantage is that the clear class-based structure from GFW can not be easily retrieved from the ORM-schema. Of course, with some in depth analyzing this is possible, but as we do not really need the class-attribute distinction for expressing the PRM, this kind of analysis is not needed. This is the ***No-Class-Needed Advantage*** or ***NCN Advantage***: ORM is capable of expressing a relational model which makes use of classes in a different way, without using classes.

Despite it is not needed, there is a way to make clear to what the class-objects are in ORM and which the attributes (or which are both), which we will call the ***Formal-Class-Definition Solution*** or ***FCD Solution***. This solution applies when you formally define your model. If we would define our RE as having the object types $\mathcal{O} = \{$Sporting Team, Player, Trainer, Country, Fan, Revenue, Prizes, Sport, Person, Age, Weight, ...$\}$, you could define subsets of $\mathcal{O}$, defining which of them are classes and which entities are the 'attributes' of these classes. For example,

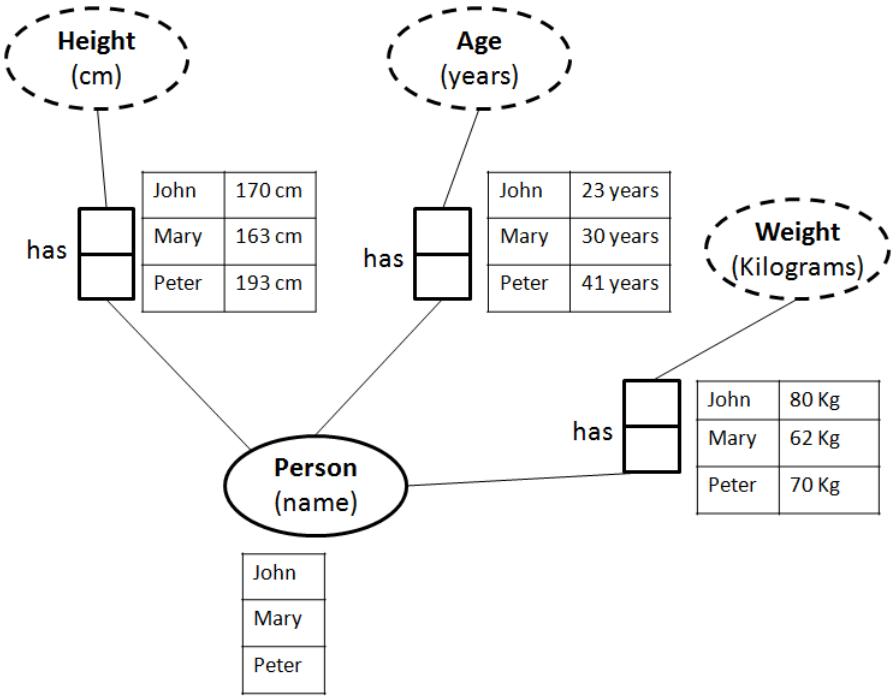


Figure 5.3: Population of an ORM schema

O_1 could be {Sporting Team} and $O_1.\mathcal{A}$ = {Player, Trainer, Country, Fan, Prize, Revenue}. However, this would mean you have to formally define you whole ORM-schema, which is a lot of work. So, when searching for a good representation of classes in ORM, graphically this can be problematic. For using PRMs in ORM, classes are not mandatory and we therefor see the loss of class representation in ORM as not important.

5.1.3 Similarity of Instances?

As with GFW, ORM needs a population of the schema to work. As with GFW, a population of an ORM schema can be denoted as $\mathcal{I}$. The contents of a population, however, differ a bit from GFW. In GFW, $\mathcal{I}$ defines over a relational schema:

1. The set of objects in the class, $\mathcal{I}(X)$
2. The value for each attribute $x.A$
3. The value of each reference slot, denoted as value y for reference slot $x.\rho$

As ORM has no classes, $\mathcal{I}$ specifies for each object (entity type and role) its instances. Value types are not specified with $\mathcal{I}$, because their value-range are assumed to be generally known. An example of a population $\mathcal{I}$ over an ORM schema is found in figure 5.3.

Example 4. The instances of Person can be defined as $I(\text{Person}) = \{John, Mary, Peter\}$. The population of, for example, the fact Person *has* Age can be defined as $I(\text{Person } has \text{ Age}) =$

$\{t_1, t_2, t_3\}$. We define population not directly, because of their dependency on the exact role they play.

If we define $I(\text{Person } has \text{ Age}) = \{\{John, 23\}, \{Mary, 30\}, \{Peter, 41\}\}$ we can not base any conclusion on this definition. For example, we can not conclude from this definition if *Peter* is the Age or the Person. Therefor, we use t_1, t_2, t_3, which are functions used as follows: $t_1(\text{Person}) = John$ and $t_1(\text{Age}) = 23$.

The differences between populations in GFW and ORM is that GFW's population not only define instances of the objects, but also the set of objects in a class and the value of reference slots. ORM has no classes and reference slots, so only defines the values (instances) of objects. Therefor, defining a population over a ORM schema is different than defining it over GFW. This is not a problem, as every modeling language defines its population in an other way, the ***Dynamic-Population-Definition Principle*** or ***DPD Principle***. It does not influence the usage of PRM in ORM and therefor the different way of populating a schema does not cause any problems for the scope of this thesis.

Now we showed that we are able to express the relational model of GFW in ORM and its populations, we can move on to expressing a PRM in ORM.

5.2 Converting PRM: from GFW to P-ORM

This section will discuss the usage of a PRM in ORM. It will take a look at if and how the concepts from GFW can be transferred to the ORM language.

5.2.1 Converting the Relational Skeleton

The first thing that is needed to make a PRM is the relational skeleton of a schema r, σ_r. In GFW, they create a new model for that, seen in the left schema of figure 3.14. An ORM-schema however, already contains all these relations between objects. And as there are no classes, it is not needed to specify the objects belonging to the classes. The only thing that is needed, is to add instances of the objects that are classes in GFW. So in short, the relational skeleton of a schema is a skeleton with only the classes, their attributes and their relations in GFW or the ORM schema without constraints and instances in ORM. If we look at figure 5.3, we see that instances can be directly be denoted in an ORM-schema. If we would regard Person as class and Weight, Age, Height as its attributes, only the values of Person would be filled in. The values of the attributes coupled to those instances would be blank (see figure 5.4. The expression of a relational skeleton in ORM and the advantages over GFW is called the ***Relational-Skeleton Advantage*** or ***RS Advantage***.

The relational skeleton than can be filled in with actual values, which in combination with the Conditional Probability Distribution (CPD) and the dependency structure defines the values for all the object types in a schema. So, we now need to know how a dependency structure and a CPD can be represented in ORM.

Converting the Dependency Structure

For the dependency structure S which is shown in the right schema of figure 3.14, in ORM you can create a similar structure. As GFW use solid arrows to illustrate dependencies,

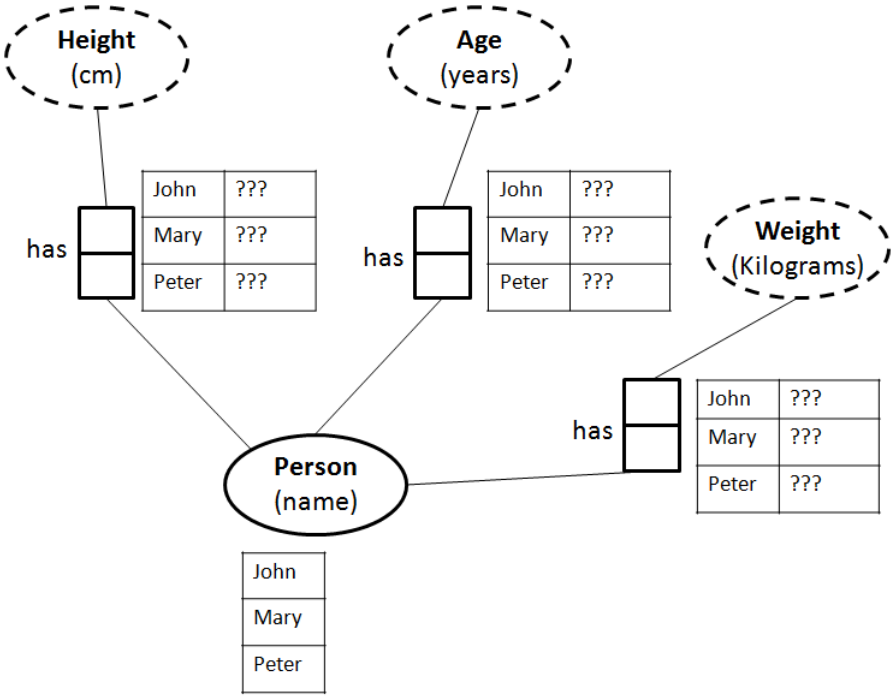


Figure 5.4: Example of a relational skeleton σ_r in ORM.

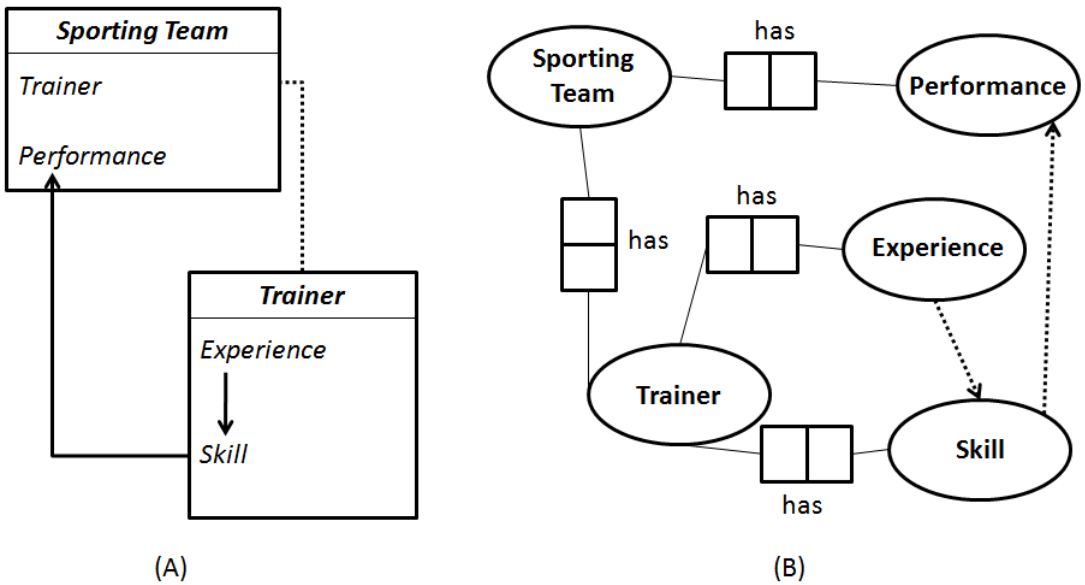
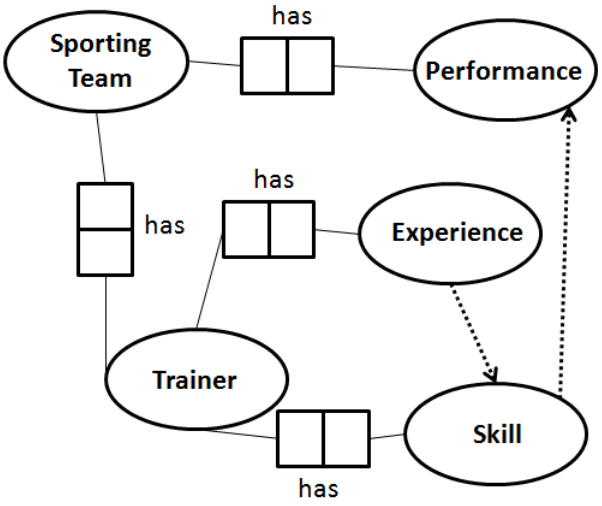


Figure 5.5: Dependencies of performance of sporting team in GFW (A) and its expression in ORM (B).



Skill *Perf.*	Bad	Good	Extr.
Low	0.6	0.39	0.01
Medium	0.4	0.5	0.1
High	0.1	0.6	0.3

Exp *Skill*	Low	Medium	High
≤ 2	0.6	0.3	0.1
≤ 5	0.4	0.4	0.2
≤ 10	0.2	0.4	0.4
> 10	0.05	0.3	0.65

Figure 5.6: A conditional distribution over an ORM schema.

we use dotted arrows in ORM. This is because solid arrows are already used in ORM for subtyping and dashed arrows for generalization. With this somewhat simple addition, we can graphically express a dependency structure in ORM. Where in GFW the dependencies are between attributes of the same class or attributes of different classes, ORM knows only dependencies between object types. These object types can be, for example entity types, objectifications and power types.

This still means that GFW make a distinction between attributes and classes and ORM does not. But, also, the dependencies are only between attributes and the dependencies themselves make no distinction between classes or attributes. Because of this, the absence of the class representation and attributes in ORM does not give any problems. Just as in ORM every object type is treated the same, for the dependency structure there are only attributes and no classes. Because GFW and ORM define the probabilistic dependencies in almost the same way, we call it the ***Dependency-Structure Similarity*** or ***DS Similarity***.

To illustrate a dependency structure in ORM, we will use a part of our running example, namely the part of the schema describing a Trainer of a Sporting Team. The Experience of a Trainer influences his/her Skill. The Skill then influences the Performance of a Sporting Team. This example can be found in figure 5.5.

Using Conditional Probability Distributions (CPDs) in ORM

Now we can define a dependency structure in an ORM schema, we will also need to define a Conditional Probability Distribution. This CPD describes which influence values of one object

type have on another value type. Just as in GFW, in ORM a simple table does the work of a CPD properly. An example of this can be found in figure 5.6. Because GFW and ORM both make use of schema-built-in tables, we name it the ***CPD-Table Similarity*** or ***CT Similarity***.

The main theory behind probabilistic dependencies uses in GFW also applies to our ORM representation. The only difference is the absence of classes in ORM. But when taking a close look at the probabilistic theory in GFW, [GFK$^+$07] make no real use of their class representation. Their theories are built around them, not with them. Because we will not actually do the calculations in this thesis, we will not discuss these theories again nor redefine them for ORM.

[GFK$^+$07] also discuss the use of aggregate functions, when an attribute depends on more than one value. This theory stays the same for ORM, only attributes are replaced with object types.

5.2.2 More Formal Definition of P-ORM

With a relational model, a way to describe probabilistic dependencies and a CPD, we are now able to express a PRM. What exactly was a PRM again?

In words:

Definition 30. *A PRM specifies a template for a probability distribution over a database. The template includes a relational component that describes the relational schema for our domain, and a probabilistic component that describes the probabilistic dependencies that hold in our domain. A PRM has a coherent formal semantics in terms of probability distributions over sets of relational logic interpretations*[GFK$^+$07].

What definition 30 in combination with the previous section shows, is that our ORM-representation satisfies this whole definition, except one point: *the coherent formal semantics in terms of probability distributions over sets of relational logic interpretations* [GFK$^+$07]. Before discussing the coherency of our PRM representation in ORM, we will first discuss the formal definition of a PRM for a relational schema [GFK$^+$07]:

Definition 31. For each class $X \in \mathcal{X}$ and each descriptive attribute $A \in \mathcal{A}(X)$:

- a set of parents $\mathrm{Pa}(X.A) = \{U_1, ..., U_n\}$ where each U_i (or parent) has the form $X.B$ or $\gamma(X.\mathbf{K}.B)$, **K** being a slot chain and γ an aggregate of $X.\mathbf{K}.B$.
- a legal CPD, $P(X.A \mid \mathrm{Pa}(X.A))$.

Because ORM makes no use of attributes and classes, the formal definition of a PRM also needs to be adjusted to that. From our research in this chapter we could deduct the following:

Definition 32. For each object $O \in \mathcal{O}$ there is:

- a set of parents $\mathrm{Pa}(O) = \{U_1, ..., U_n\}$, where each U_i is an other object out of $\mathcal{O}$ or an aggregate γ if O depends on more than one value of an object type.
- a legal CPD, $P(O|\mathrm{Pa}(O))$.

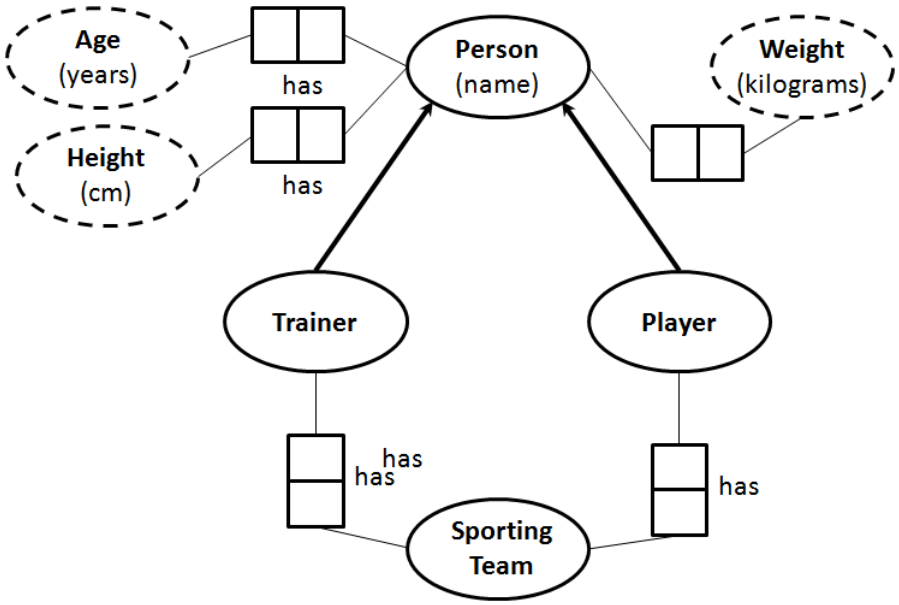


Figure 5.7: A simplified part of our RE, showing subtyping in ORM.

You can conclude from this definition that every object type should have a dependency attached to it. In practice this is not the case, not every object type has a role in the dependency model. But to make the definition of a PRM easier, [GFK+07] abstracted from this notion and so did we.

The redefinition of the formal definition for PRMs of GFW to make it usable for ORM is called the ***Formal-PRM Redefinition*** or ***FP Redefinition***. With this, the definition that is given for GFW is also usable for ORM.

5.2.3 Class Hierarchy: from PRM-CH to P-ORM-CH

The usage of class hierarchy (CH) is something that ORM is familiar with. As could be seen in the previous chapter, ORM is able to explicitly define subtypes, the name for class hierarchies in ORM. In GFW, a separate tree structure was needed to show the class hierarchies for their model (see figure 3.18). In figure 5.7 you can see that ORM can add subtypes in the relational model itself.

In figure 5.7, we see a simplified part of the RE-domain. A sporting team has a trainer and a player. Both Player and Trainer are subtypes of Person, inheriting the relations Person has with other object types. In the figure, we see a person having an age, a weight and a height. The trainer and the player automatically inherit the relations with these three entity types.

[GFK+07] have to create more than one schema to model inheritance in their framework. They not only need their relational model with a PRM, but also a separate model which shows the hierarchies and dependencies for that relational model. Figure 5.8 shows the model with class dependencies and hierarchy from [GFK+07]. On first sight, this model works fine. But, when taking a closer look, it does not become clear if Action Movie and Documentary are subclasses from the class Movie. When representing this model in ORM, we loose the ability to represent the classes, but we can show subtyping, the dependency model and the relational

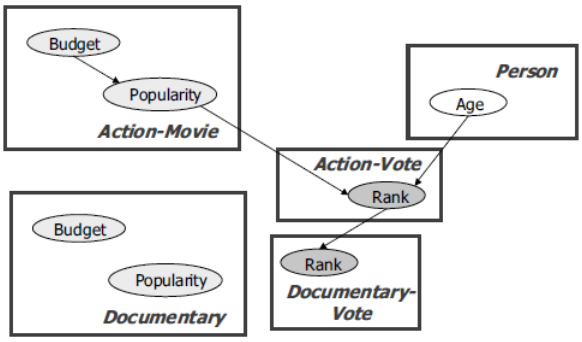


Figure 5.8: The dependencies for action movie votes and documentaries from [GFK+07].

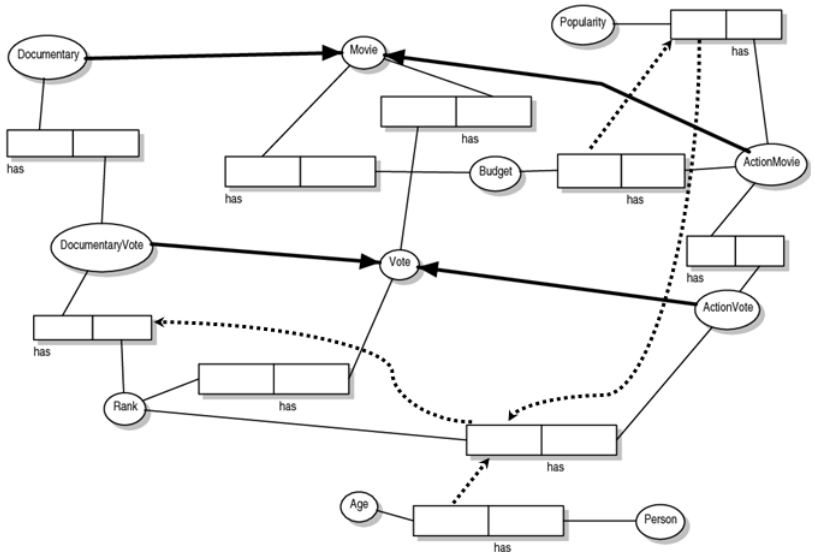


Figure 5.9: The Movie domain from [GFK+07] in ORM.

skeleton all in one model. The possibility to express subtypes in the relational model shall be called the ***Subtyping Advantage*** or ***ST Advantage***.

Because classes were not important for their PRM-theory and class hierarchies are, this could give ORM an advantage over GFW. According to [GFK+07] with class hierarchy, we were able to express relation that could not be expressed without CH. Figure 5.9 shows figure 5.8 expressed in 'step 2' ORM (everything up to step 2 from section 4.3).

What we see in figure 5.9 are the classes from the GFW-version, depicted as entity types and with relations between them. Also, Person and Age are not connected with normal ORM-relations to any other entity type in the schema, which normally means they should be left out of the schema. However, they have a dependency relation with Rank and are therefor part of the schema.

In the ORM schema, all the relations GFW try to express in two schemas are tried to be expressed in one. When comparing this schema to the schema 5.5.B, the dependency arrows are drawn differently. The reason for this is that we wanted to show the class hierarchy. The strength of CH in GFW was that you could isolate a set of values, for example Action Vote

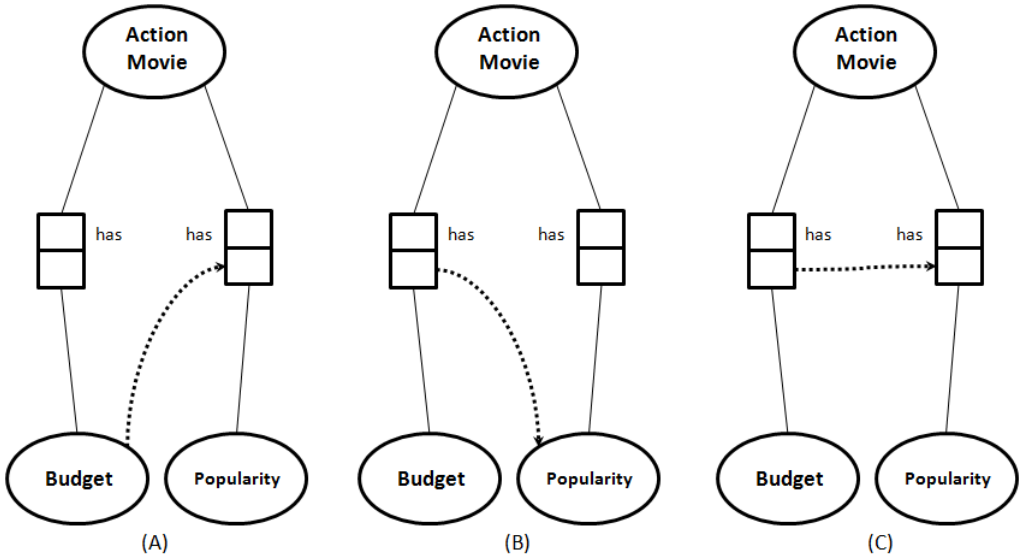


Figure 5.10: Three possibilities of role dependencies in ORM.

and say the ranks belonging to the action votes influence the ranks of the documentary votes.

As you can see in the ORM schema in figure 5.9, we did create separate entity types for action votes and documentary votes, both subtypes of the entity type Vote. Here, an advantage of using classes in a relational model becomes clear: the attribute Rank in Action Vote is an other attribute than Rank in Documentary Vote. To represent this structure in ORM, we can do two things:

1. ***Isolate-By-Subtyping Solution*** or ***IBS Solution*** Create two different entity types, Action Rank and Documentary Rank, the first belonging to Action Vote, the second to Documentary Vote. Both Action Rank and Documentary Rank are subtypes of the entity type Rank. This solution has been used for representing the two kinds of Vote in the schema, Action Vote and Documentary Vote.

2. ***Isolate-With-Roles Solution*** or ***IWR Solution*** The other way to represent this structure is the one used in the scehma. We make use of the expressiveness of relations in ORM. In a relation, there are roles, containing those values of a entity type playing a role in that specific relation. As said, in GFW you could isolate values of certain attributes by using CH. In ORM, the same thing can be done by making smart use of suptyping and roles. Because we created two subtypes for Action Vote and Documentary Vote, we have two separate facts: Action Vote has Rank and Documentary Vote has Rank. Both the roles played by Rank in these facts contains only values belonging to either Action Vote or Documentary Vote. We can therefor attach the dependency arrows to the roles and not the entity types itself.

The big advantage of the second option is that we do not have to add a new entity type with subtypes for every dependency relation we want to make to isolate certain values. The more object types you add to an ORM schema, the more relations there are and the complexer the schema gets. And the more object types and relations there are, the more the schema

becomes unclear and unreadable. Also, when attaching dependencies to roles, you can isolate dependencies to certain parts of the schema. An example from the movie-domain: if the dependency arrow was not between Age and the role Rank plays in Action Vote has Rank, but between Age and the entity type Rank, all rankings would be influenced by the age of a person. It could be the case that for documentaries the age of the person does not matter when ranking it. By attaching the dependency to a role, you make the dependency only valid for role, not the whole entity type.

There is a catch however. Not every dependency arrow can be drawn from one role to another. There are cases where all values of a entity type can influence the values of an other entity type or role, or a role influences all the values of an entity type. So, it is always needed to be certain what kind of dependency you are trying to add. This is the ***Dependency-Scope Problem*** or ***DSc Problem***. We will not discuss an in depth analysis or correctness of these dependencies in this thesis, for the same reasons as coherency is not discussed (see the previous section). We focus on the graphical possibilities of ORM, not on the possible problems of formal correctness that are behind them. Figure 5.10 shows the three other possibilities, with the already discussed entity-type to entity-type dependency as fourth one. The other three are: A) from entity type to role, B) from role to entity type and C) from role to role.

5.2.4 Notion on Coherency of P-ORM

As seen in section 3.4, GFW created a theory for satisfying acyclicity in their dependency model. To speak of coherency in P-ORM, we also need a theory like the one of GFW. In this thesis, the coherency of P-ORM (combination of ORM and PRM) will not be discussed. We focus on the practical part, showing the possibilities of graphical representations of PRMs in ORM. Making P-ORM formally coherent is something for future research.

5.2.5 Concluding

What we showed in this section is that it is possible to express GFW in ORM. Because of the many found problems, solutions and similarities we have put a summary of our findings on expressing GFW in ORM in table 5.2.5. The first column describes the concept from GFW, the second column states if it is usable or expressible in ORM and why. The third column gives the alternatives or solution ORM has for that concept or why it is not a problem that ORM can not express it. We use the terms indicated with the bold and italic font style found throughout this first section.

5.3 Possibilities of P-ORM

In this section we want to discuss some the advantages ORM can give over other relational languages which can be used in combination with PRM. Because ORM is one of the most expressive modeling languages, it is possible to express structures that other modeling languages can not express properly. Because this thesis only discussed three frameworks with their own relational language, this section can only compare ORM to those three. As we used

GFW	Possible in ORM?	Solution in ORM
Relational Model		
Classes & Attributes	no, CR Problem	NCN Adv., FCD Sol.
Class - Attribute Relation	yes, IR Translation	-
Reference Slot (Chain)	no, CR Problem	NCN Advantage
(Definition of) Population	no, CR Problem	NCN Adv., DPD Princ.
PRM		
Relational Skeleton	yes, RS Advantage	-
Dependency Structure	yes, DS Similarity	-
CPD	yes, CT Similarity	-
PRM Definition	no, CR Problem	FP Redefinition
Coherency of PRM	not discussed	not discussed
Class Hierarchy		
Class Hierarchy Tree	yes, ST Advantage	-
Isolation of attribute values	no, CR Problem	IBS Solution, IWR Solution
Coherency of CH-PRM	DSc Problem, not in-depth discussed	not discussed

Table 5.1: (Im)possibilities of the usage and expression of concepts from GFW in ORM.

GFW this whole chapter and we think this is the most workable framework for PRMs, we will try to show some situations in which ORM can have an advantage over GFW.

We will first focus on structural advantages, discussing three concepts which are often used in ORM. We will then also brainstorm a bit about the usage of constraints. Because the focus of our thesis is not really on formality and constraints have a big deal of formality behind them, we just want to think a bit about the possibilities for P-ORM they could give. The last section will summarize this chapter of our thesis.

5.3.1 Examplary Structures of ORM: New Possibilities in P-ORM?

The main advantage ORM has over other modeling languages is that it can express almost every relation, giving it the ability to express structures other modeling languages can not express. GFW has no explicit expression of relations as ORM knows them, limiting themselves only to explicit relations between reference slots and classes. In this section, we will show three structures that are an example of the expressiveness of ORM. We will try to match these three structures to their possible counterpart in GFW and see if one of them has an advantage over the other. The three structures we will discuss are: facts (or relations) that contain more than two roles, objectifications and power types. Figure 5.11 shows these three structures: A) *ternary fact type*, B) *objectification* and C) *power type*. After discussing these three examples we will also discuss the usage of constraints in ORM.

Using Ternary Fact Types in P-ORM

As discussed in the previous chapter, facts in ORM can have as many object types connected as you want. Of course, if more objects are connected to a fact, the fact becomes more

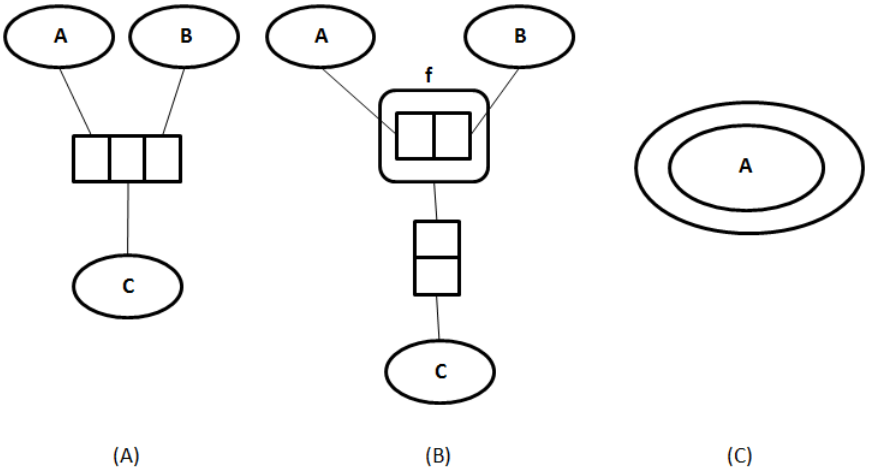


Figure 5.11: Three typical structures in ORM.

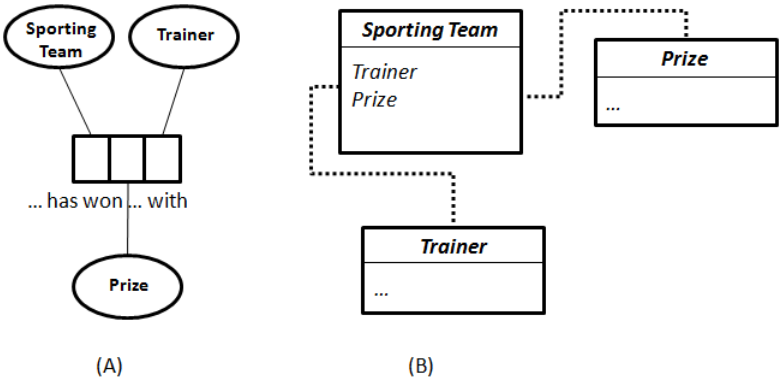


Figure 5.12: Ternary fact type in ORM (A) and its possible representation in GFW.

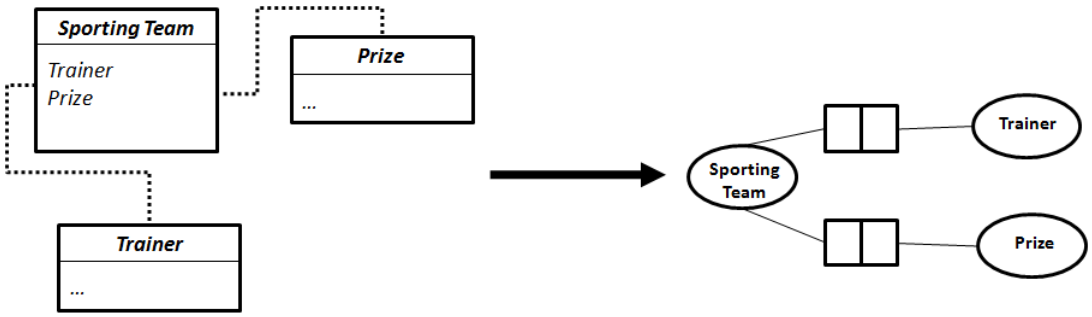


Figure 5.13: Conversion of supposed ternary fact type in GFW to ORM.

complex. In ORM, binary and ternary fact types are the most used. In GFW, we saw they can represent binary fact types. From the university domain, we saw all kinds of binary relations, implicit to the class hierarchy [GFK+07] built. But maybe we want to take these relations a bit further, expressing relations where three or more objects are involved. This is not natural for the relational model in GFW, so we ask the question: is it possible to express a ternary relation in GFW?

When we want to express something from the RE like Sporting Team has won Prize with Trainer it becomes hard to see how this relation can be expressed in GFW. First of all, in ORM this would be a simple ternary fact type. In our ORM-schema for the RE (figure 4.6) we chose to objectify Prize, but to make the example less complex, for now just connect three entity types to a ternary fact type. When expressing a fact in GFW, we first need to identify classes and attributes. Sporting Team, Prize and Trainer all three are classes, with Prize and Trainer also being reference slots in the class Sporting Team. Figure 5.12 shows both the ORM as the GFW expression.

Does the expression in GFW represent what we could express in ORM? Well, not so clearly. If we would apply our translation of relations from GFW to ORM, two facts are derived from the class Sporting Team: Sporting Team has Trainer and Sporting Team won Prize, see figure 5.13. We can not directly deduct the ternary ORM-fact, because Trainer and Prize are not directly linked. This is what we will call the ***Ternary Expression Problem*** or ***TE Problem***. When we populate both ORM schemas (the *ternary* (5.14.A) and the *double-binary* (5.14.B)) we can see that the *double-binary* schema allows other populations as the *ternary* does.

This can be deducted as follows: first we take the population of schema A as the correct population (as this is the original ternary fact type). We translate the ternary fact to GFW (not explicit in this figure) and translate this GFW-translation back to ORM, which gives us schema 5.14B. The only way to deduct the ternary fact combinations of A out of B is to combine the two binary facts. We can than deduct the following facts:

1. Barcelona has won CL with Guardiola
2. Barcelona has won Primera Division with Guardiola
3. Barcelona has won CL with Cruyff
4. Barcelona has won Primera Division with Cruyff

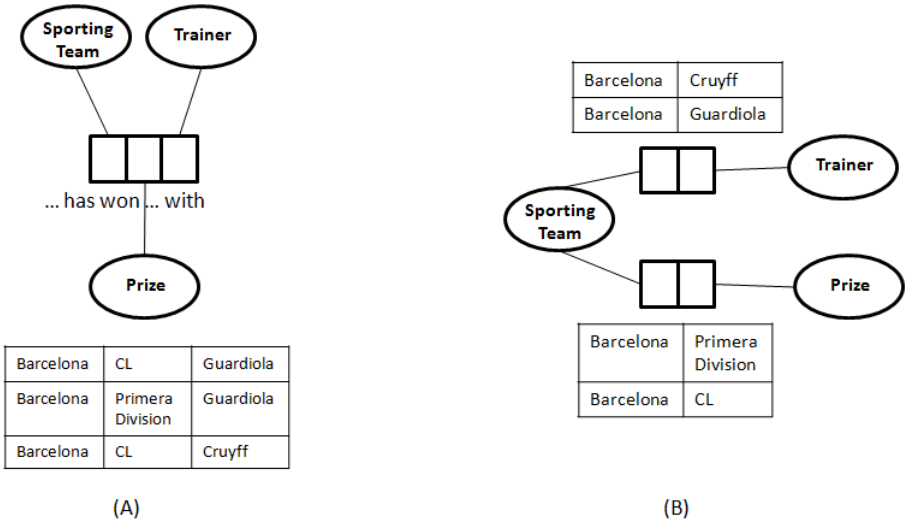


Figure 5.14: The ternary (A) and binary (B) schemas with population.

The first three facts are the same as those from schema A. But fact 4 is a non existing fact in the population of B. So, when expressing a ternary fact type in GFW, we need to combine two binary fact types to one fact. With this combination, sometimes a combination can exists which was not in the original population. We call this problem the ***Population-Combination Problem*** or ***PC Problem***.

So, GFW is not able to express a valid ternary fact type in their framework, because of the *TE-* and *PC Problem*. This gives ORM a structural advantage over GFW when expressing facts with more than two roles (or relations containing more than two classes or attributes).

For dependency relations, ternary fact types are not really able to express something more than GFW could without them. Because we can not attach dependencies to a fact type, we first need to objectify the fact type to attach dependencies to it. In that way a ternary fact type is needed to add an objectification to, but it does not give a direct advantage for the PRM, making it ***PRM Indifferent***. So, for a PRM it does not really matter if you can express ternary fact types or not. But for expressing a relational model and its population it makes a real difference.

Using Objectifications in P-ORM

Objectification is the possibility in ORM to make an object type out of a fact that can have relations with other object types itself. This means that the combination of roles from the fact are the new object type. The ability to do is directly enabled by the expressive way in which ORM treats relations. Figure 5.15 shows an objectification of a binary fact type f, which has a binary relation with the entity type C. It also shows a possible dependency relation with corresponding CPD.

GFW has a ***Objectification-Expression Problem*** or ***OE Problem***, meaning they have no mechanism to express objectifications. There is no way make a class or an attribute out of a relation in their relational model. A small comfort for them: in almost no other relational modeling language there is this possibility. The facts objectifications represent in ORM how-

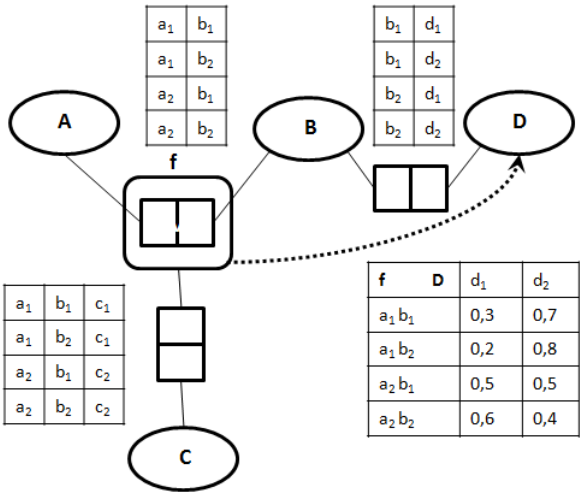


Figure 5.15: An example of objectifcation in ORM and a PRM over it.

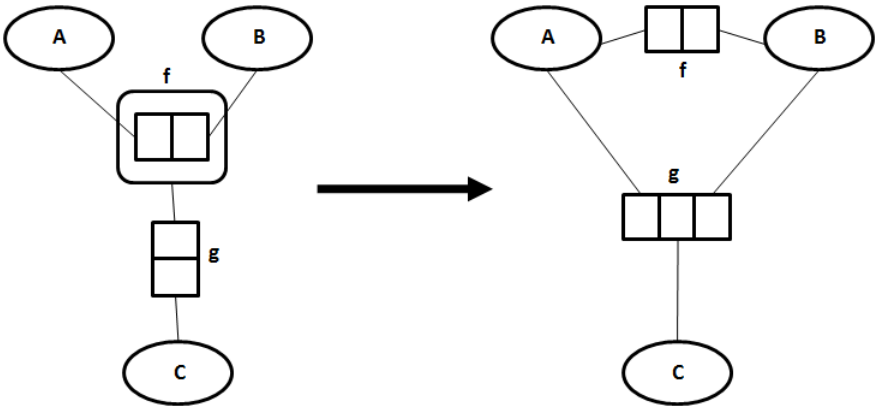


Figure 5.16: An example of how an objectification can be removed in ORM.

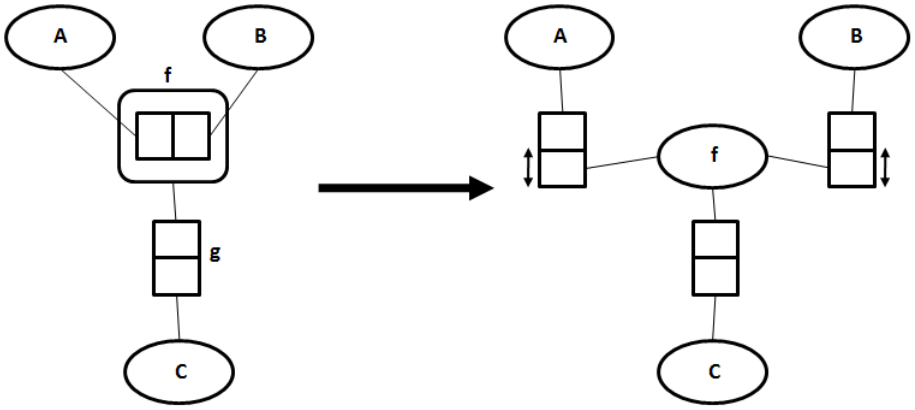


Figure 5.17: The other way of removing an objectification in ORM.

ever can also be expressed by other constructions, making objectifications not a mandatory element for expressing the domain. Figure 5.16 shows such a construction, eliminating the objectification. Because we don not want to loose any facts, fact f and fact g of the left schema are also drawn in the right figure and not put together in the ternary fact g. When removing the objectification, you can see it becomes a ternary fact type. For expressing a ternary fact type in GFW, we already saw the *ternary-expression* and *population-combination* problems occur. We shall call the whole objectification removal problem the ***Objectification-Removal Problem I*** or ***OR Problem I***.

There is another way to remove an objectification, which figure 5.17 shows. The idea is that entity type f contains the unique combinations of A's and B's which were first found in fact type f. Representing this solution in GFW gives is yet again the *binary-combination problem*: we can not restrict the values which we combine with each other. Also, in GFW there is no way to express uniqueness constraints. Therefor, this objectification removal is not an option and we shall call this the ***Objectification-Removal Problem II*** or ***OR Problem II***.

As you can see in the figure 5.15, expressing a CPD over an objectification works the same as it does for a normal entity type or role. Still, the ***CPD-Complexity Disadvantage*** or ***CC Disadvantage*** can occur: the CPDs for objectifications can become somewhat bigger, because populations of fact types can contain more values than the object types they are connected to. It is good to keep this in mind when creating CPDs, because when the larger a CPD is, the more complex they are.

As we discussed above, an objectification can be easily removed in ORM. But this gives us some problems with the dependency structure of the newly created ORM schema. Until now we were only able to express dependencies from and to one entity type and one role. With an objectification, we could take two or more roles together and attach dependencies to it. Figure 5.18 shows that when removing the objectification, it graphically becomes harder to express the same dependency. We now hooked the two roles together, but this is not as clear as the objectification itself is. Even more when we would also add constraints on the role, the whole outlook of fact f would become messy. We can say that an objectification has an ***Objectification-Dependency Advantage*** or ***OD Advantage***, letting it express probabilistic dependencies attached to more than one role in a fact more easily.

In short, objectifications give an extra mechanism to express relations in ORM and they

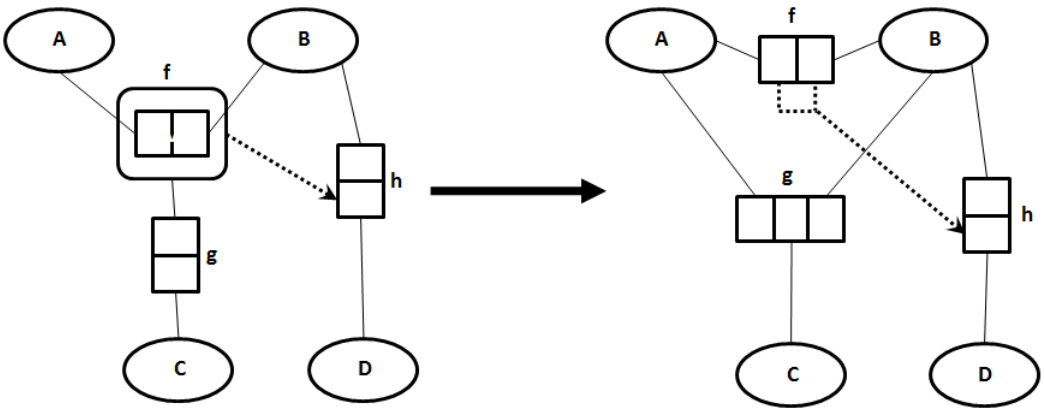


Figure 5.18: Removing objectification and its dependencies in ORM.

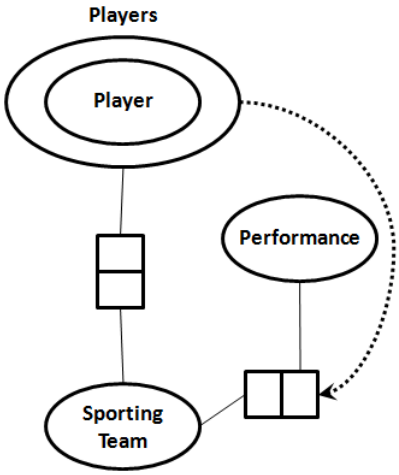


Figure 5.19: Schema with power type and attached dependency relation.

have no counterpart in GFW or in most other modeling languages, which we called the *OE Problem*. This gives ORM an extra way to express aspects of the universe of discourse and probabilistic dependencies that are connected to it. Still, the part of the domain an objectification models can also be expressed by other constructions, as figures 5.16 and 5.17 show. But these expressions gave us the *OR Problems I & II*.

When adding dependencies, objectifications can express something more than is possible in GFW: it can add a combination of values to the dependency structure, as figure 5.15 shows. When removing an objectification, these dependencies are graphically harder and less clear to express in ORM itself. So, objectification have an *OD Advantage*. In GFW, there is no valid way to express an objectification, an alternative structure for an objectification or the dependencies which can be attached to an objectification.

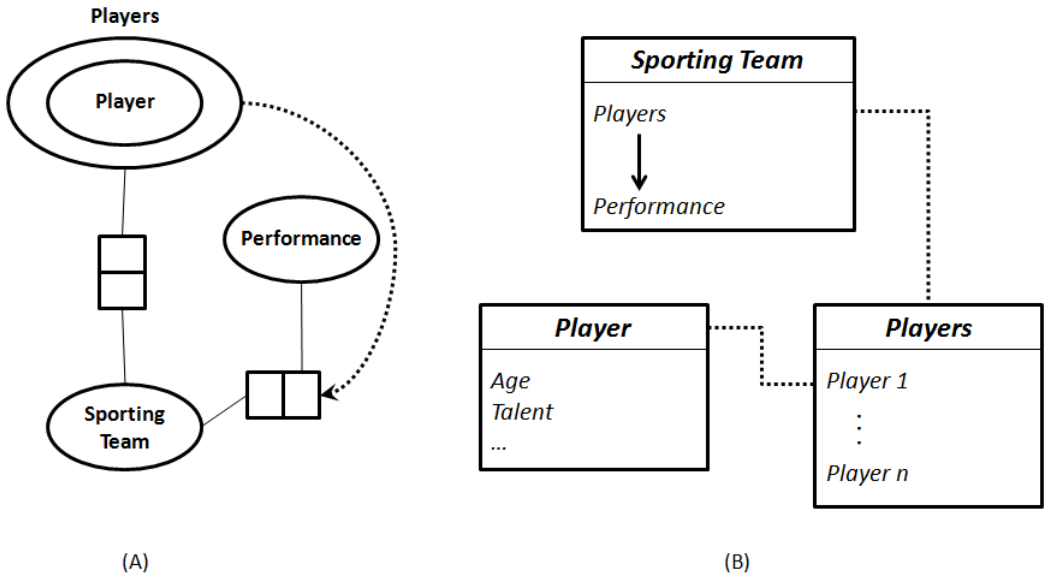


Figure 5.20: Power type and dependency relation in ORM (A) and its representation in GFW (B).

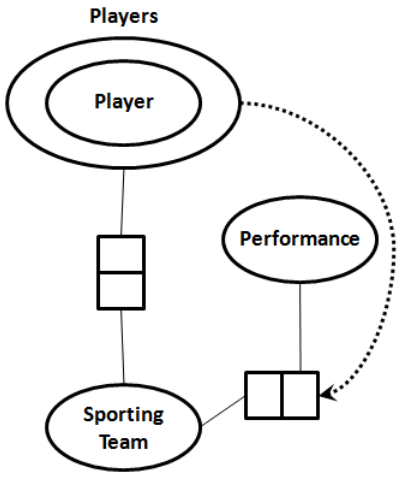
Using Power Types in P-ORM

Power types are an construction in ORM to make it possible to create a set of instances of an entity type. In our RE, we have the entity type Player. When we create a power type over it, we can create sets of players, named Players. Figure 5.19 shows an example of a schema containing a power type and a dependency relation it plays.

The power type is a very powerful mechanism to create sets of a variable amount of instances of the same entity type. As with almost all other object types in ORM, constraints can be added which, for example, limit the amount of instances a set may contain. In all other frameworks we discussed in this thesis, a variable amount of instances of something was problematic. [KP97] explicitly mentioned this problem, but in the frameworks of [BW00a] and [GFK+07] there also was no solution for this. Of course, a power type is always limited to one entity type. This makes it a limited solution for the problem. The problem of these frameworks with an unknown or varying amount of instances is called the ***Varying-Instance Problem*** or ***VI Problem***.

But could GFW construct something like a power type? Well, the closest they could get is making a class named Players containing a variable amount of attributes belonging to the class Player. Figure 5.20.B shows a representation of the schema in GFW. Still, there is no mechanism in GFW to represent a varying amount of attributes from the same class or put attributes or create a set of instances from an attribute. The inability of GFW to express a valid power type is what we call the ***Powertype-Expression Problem*** or ***PE Problem***.

We showed that power types can be functional for creating sets of instances from the same entity type. But is it also functional for P-ORM? If we would take the example from figure 5.19, a possible interpretation of the dependency between Players and Performance is that a certain combination of players in a team influence the performance of that team. Take



Players Perf.	Bad	Good	Extraordinary
Messi, Xavi, Iniesta	0,1	0,5	0,4
Messi, Pedro, Iniesta	0,2	0,6	0,2
Pedro, Iniesta, Villa	0,4	0,5	0,1
Pedro, Messi, Villa	0,2	0,7	0,1

Figure 5.21: Example of a CPD with a power type.

ORM	Expressible in GFW?	Alternative in GFW?	Function in P-ORM
Ternary Fact	no, TE Problem	no, PC Problem	PRM Indifferent
Objectification	no, OE Problem	no, OR Problem I & II	CC Disadv., OD Adv.
Powertype	no, PE Problem	no, VI Problem	SD Advantage

Table 5.2: Concepts from ORM, expression in GFW and (dis)advantages in P-ORM.

football for instance, where Barcelona has a team with Messi, Xavi and Iniesta in it. Such a combination of players would likely give you a good performance and sometimes even an extraordinary performance. When lesser players are included instead of Messi or Xavi, the probability of a good performance will be lower. A CPD for this example is found in figure 5.21. This gives ORM the advantage to create dependencies on sets of instances, something GFW can not do. This is called the ***Set-Dependency Advantage*** or ***SD Advantage***.

As you can see, the usage of a power type in ORM can give some advantages: it enables you to create sets of instances from an entity type and you can vary the amount of instances in the sets (*VI Advantage*). Still, power types only give a solution for a varying amount of instances from one entity. In P-ORM, you can add dependencies to a certain combination of instances (*SD Advantage*) as our example above showed. This is something which GFW lacks. We tried to create a schema in GFW which would have the same mechanism as a power type in ORM, but had to conclude this was not possible (*PE Problem*).

Concluding

In this section we looked at what kind of advantages some structures typical for ORM could give ORM an advantage over GFW in both the relational model as the PRM. As we did in the previous section, we will summarize the section with a table, found in table 5.2. The first column states the concept from ORM, the second if it is (in)expressable in GFW and why (not). The third column looks at the alternatives GFW has for the concepts from ORM and the fourth column gives the (dis)advantages the concept could have in P-ORM.

When looking at this section and the table, we must conclude that ORM is able to express

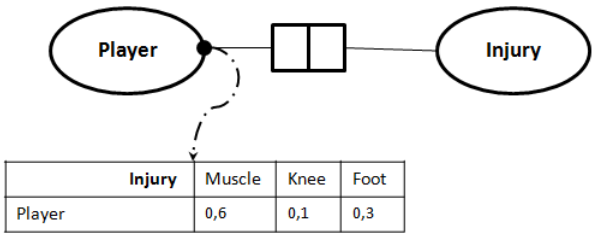
more relations and dependencies than GFW can. GFW is not able to express all the concepts ORM has. Two of the three concepts we discussed gave new possibilities in the P-ORM schema. Still, more expressiveness also gives more complexity.

5.3.2 The Possibilities of Constraints in P-ORM

This section is more of a brainstorm, discussing the usage of constraints in P-ORM. Because the is no mention of constraints in GFW and they have no explicit relations to which constraints can be added, we already need to conclude that constraints are (almost) impossible to express in GFW. Because constraints are on of the strengths of using ORM, it is very interesting to take a look at what advantages constraints could give us in the P-ORM, the combination of ORMs and PRMs. Because the scope of this thesis is on the graphical aspects of PRMs (and P-ORM), we take some constraints found in ORM and see if they could do something with the probabilistic dependencies found in a PRM. We will no discuss the correctness of the constraints in P-ORM or the formal meaning of them on populations of P-ORM schemas and their CPDs.

What we will do in this section is show some models which use constraints with attached probabilities. The focus will be on mandatory role constraints, but we will also make use of uniqueness and exclusion constraints.

Mandatory Role Constraints and P-ORM



Injury	Muscle	Knee	Foot
Player	0,6	0,1	0,3

Figure 5.22: Using a mandatory role constraint as a *gatekeeper*.

The mandatory role constraints (MRC) in ORM are put on roles, forcing that all instances of an entity type play a role in that fact (see figure 5.22). You can also put them between two roles belonging to the same entity type of different facts, forcing that the joined set of instances of that role is the whole population of that entity type (see figure 5.23).

The first usage of a MRC in P-ORM is as ***Gatekeeper***, as shown in figure 5.22. The idea behind the schema is as follows: a player sustains injuries in his sporting career, shown with a binary fact we name Injured. There are many kinds of injuries, but to keep the example simple there are only three: muscle-, knee- and foot injuries. Because all players in this example are very fragile, they all sustain injuries (so every player has to play the role in the fact Injured. The *gatekeeper* idea is: every player comes past the gatekeeper, which labels them with an injury. Every injury has a chance of occurring, with the total of injury-probabilities being 1.

We could also add an uniqueness constraint to the role of Player, stating that every player can only sustain one injury. In this schema however, a player can have an endless list of injuries, even of the same one.

There are, of course, some problems with this example. First of all, in the real world not every player gets injured. In that case, we could not use a MRC because not every player plays the role. Also, not every entity type has a limited population. If all probabilities need to add up to 1, we need some kind of mechanism to let this work for every entity type. Categorizing the population of an entity type could be such a solution. Implementing the *gatekeeper*-idea could be useful, but as said above: we are just brainstorming so it is nowhere near usable.

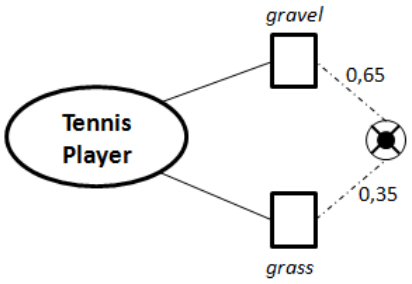


Figure 5.23: Using the mandatory role constraint as a *divider*.

But there are other usages of a mandatory role constraint. Figure 5.23 shows the usage of a MRC as a ***Divider***. The idea behind the schema is as follows: all tennis players have a favourite surface to play on. To keep the schema simple, all tennis players either favor grass or have a gravel as favorite. The constraint we see on attached to the roles is a disjuncitve mandatory exclusion role constraint, a combination of an exclusion constraint and a disjuntive MRC. It means: either you favor grass or you favor gravel, but never both. The *Divider*-idea is that you can attach a probability to both roles, saying that there is a 35% chance you favor a grass surface and 65% chance you favor a gravel surface.

The divider-idea is very simple, but when attaching it to binary facts is harder. Because we want to keep it simple in this section, we used two unary facts, which are booleans: if the instance plays the role it is *true*, if it does not it is *false* and the fact does not exist. The *Divider* has no real direct problems, still usage in a real P-ORM schema still needs to be researched.

Usage of other Constraints in P-ORM

As ORM has more constraints than the MRC, there are more possibilities of the usage of probabilities in combination with these constraints. What we want to point out is that ORM is one of the most expressive modeling languages that exist today and have a very extensive range of constraints to use. By enabling the usage of constraints in the PRM, ORM can become a very interesting choice as relational model for PRM models.

Concluding

With this section we tried to show some possibilities with the combination of constraints and PRMs. We looked at MRC and how they can be used in a PRM. We found two possible appliances: the *Gatekeeper* and the *Divider*. Of course, there are more possibilities also with other constraints. Because ORM has as strength the use of constraints, using them in PRMs can give ORM an advantage over every other modeling language.

5.4 Concluding

In this chapter, we showed that the PRM framework, created by [GFK+07], GFW, can be expressed in ORM. We encountered some problems with direct translations in the relational model of classes (*CR Problem*) and finding explicit relations in GFW that could be created to facts (*IR Translation*). But for every concept out of GFW, we could find a representation in ORM or loose the concept in ORM without consequences (*NCN Advantage*).

The PRM that was in GFW was also easy to translate, as most of the dependency relation could almost be translated one on one (*DS Similarity*). Even when lossing the class representation, ORM is capable of creating PRMs with the same expressiveness as GFW, with being even easier to use when it comes to Class Hierarchies (*ST Advantage*). Because the scope of this thesis is on the graphical representations, we did not discuss the coherency of the P-ORM model.

After mapping GFW to ORM, we could look at the possible advantages the expressiveness of ORM could give over GFW and to PRMs in general. We looked at fact types containing more than two roles, objectifications and power types. For all three GFW had trouble representing them (*TE-*, *OE-* and *PE Problems*). Also, in case of the objectifications and power types, it gave the possibilities to express dependencies which were not possible in GFW (*OD-* and *SD Advantage*).

For constraints, we saw that they could also be used in the probabilistic aspect of P-ORM. We showed the *Gatekeeper-* and *Divider* idea for MRC. Because MRCs are not the only constraints in ORM, there are many more possibilities of the usage of constraints in combination with the PRM. Because this thesis is focused on graphical representation, we did not go into formalities of the constraints, making it just some kind of a brainstorm of ideas about constraints.

On the whole, we showed that ORM can give advantages over GFW both the relational model as the PRM over the relational model. Still, more expressiveness also gives more complexity, which is not always a good thing.

Chapter 6

Conclusion and Discussion

6.1 Summary

In this thesis we tried to find an implementation for a PRM in the ORM modeling language. We started out by looking at the scientific literature and analyzed the history of the PRM models. A PRM basically consists of a relational model and a probabilistic model attached to it. Object Oriented Bayesian Networks (OOBNs) by [KP97] were the first attempt at creating a combination of a relational model and a probabilistic model, which was based on Bayesian Networks. [Pfe99] were the first to speak of PRMs (OOBN and PRM both refer to the same concept). [KP97] and [Pfe99] inspired the frameworks [BW00a], [LB01] and [GFK+07] created. Where [BW00a] edited the framework of [KP97], enabling top-down representation and making the framework less complex, [GFK+07] mapped PRMs to a database oriented relational model. [GFK+07] inspired [SEJ10] for their PRM framework for security analysis, based on UML Class Diagrams. Besides [SEJ10], in 2010 [TWG10] updated some inheritance flaws in PRMs which were not tackled by the earlier literature.

We then dug deeper into three important PRM frameworks from the mentioned literature: the first OOBN/PRM framework of [KP97], the top-down approach of PRMs by [BW00a] and the more practical applicable and database oriented approach of [GFK+07]. Besides that, we introduced our Running Example (RE), which we used to explain concepts from the different articles. The most important notion of this chapter is that [KP97] implement a rather complex framework, [BW00a] simplify this framework and add a top-down representation. [GFK+07] yet again simplify the usage of PRMs and dig a bit deeper into the probabilistic theories behind the framework. Their framework used a relational model comparable to UML CD, which is related to ORM schemas This made the framework of [GFK+07] the most usable framework for a mapping to ORM.

Before mapping PRM to ORM, we needed to explain the language ORM. We discussed what the theories behind the language were and which graphical representation they used. Also, we modeled our RE in ORM, showing the workings of this language. As the last part we motivated the usage of ORM over an other modeling language. The two most important reasons were that ORM has not yet been used for representing PRMs, while being a known language in many industries and the other reason was that ORM is one of the most expressive modeling languages. This makes ORM a language which can support almost every domain and can also be more expressive than UML CD. For these two reasons, ORM was the chosen

language for a combination with PRMs.

Chapter 5 saw us discuss the mapping of the framework of [GFK+07], GFW, to ORM. The name we use for this combination is P-ORM. The first section showed if ORM could express all the concepts from GFW and what problems this would give. The major issue of this expression was the lack of class representation in ORM (*CR Problem*). Still, also for this issue solutions were found (NCN Advantage, FCD Solution). Mapping the PRM from GFW to ORM also had its problems, but here every problems with the representation could be resolved (*IBS & IWR Solution*).

The second part of chapter 5 was used to see if ORM could give some advantages over GFW. Because ORM is more expressive in many ways, we looked at three exemplary concepts of ORM: *ternary fact types, objectifications* and *power types*. We found that in GFW, representing these three concepts is hard, if not impossible (*TE-, OE-, PE Problems*). An objectification and power type could give some new possibilities when attaching probabilistic dependencies to them, constructs that GFW is not able to express (*OD-* and *SD Advantage*).

We concluded chapter 5 with two examples of usage of constraints in P-ORM, where we used a *mandatory role constraint* and a *disjunctive mandatory role constraint*. We showed that the first could be used as *Gatekeeper*, while the last could be used as *Divider*. Because the focus of this thesis was not on the formal side of the relational models, we just hinted at the usage of constraint, never really formally validating them.

6.2 Evaluation of Subquestions

In our introduction we stated that the goal of this thesis was to look at existing PRMs and see if a combination with ORM was possible. The focus of this thesis was on the graphical representations, not the formality behind the models.

In this section we will evaluate our four subquestions and see if we researched what we said we would research. Also, we will point out some interesting insights we gained in this thesis and problems that we found during our research.

6.2.1 What is a PRM?

We defined that as output for this subquestion we would find a global definition for PRMs and also give a short overview of the articles written about PRMs and the ideas in them.

What we found as definition for PRMs was a definition [GFK+07] give in their article. This definition was useful throughout the thesis, it even gave us the possibility to test P-ORM with. More over, the definition applied to every article we discussed.

When we looked at the difference between OOBNs and PRMs, we gained an interesting insight: PRM and OOBN were both used to refer to the same thing. Eventually, only the term PRM was used, making it the official term for what we now know as PRM.

Another insight we gained from this subquestion was that not many researchers have applied modern modeling languages on PRM. Most frameworks used their own created relational language. GFW was the first to try something that looks like a database representation and [SEJ10] used this as base for their security based UML PRM framework.

If we look at the goals we set for this subquestion, we fulfilled them without real problems. The scientific literature about PRMs is not that extensive, with a lot of articles dating from the

beginning of this millennium.

6.2.2 What does a PRM Framework look like?

In this subquestion the goal was to find concrete examples of PRM frameworks and describe them. We also wanted to introduce our RE here, which used throughout our thesis.

The introduced RE was very useful for the whole thesis. By interpreting the RE not to strictly we could use it throughout the thesis as example for many concepts. Because the sports domain is very big and known the used examples are also understandable for almost every reader.

Besides the RE, we described three frameworks. We wanted to show the development and diversity of PRMs since 1997. Because [KP97] was the first to create a PRM framework, we thought it was needed to first show their implementation. Problematic with this was that they lacked a bit in graphical examples of their framework. This meant we needed to create a lot of examples our selves.

We described the framework of [BW00a] because a lot of later literature refers back to them. Also, they compared their framework to the framework of [KP97], giving us the chance to show the evolution of PRMs in those years. An interesting insight we gained from [BW00a] is that [KP97]'s framework was more complex than it needed to be. Even without their top-down representation, [BW00a]'s framework was easier to use. The comparison-table [BW00a] gave in their article (table 3.1) helped us a lot in gaining insight in the complexity of [KP97]'s framework and the less complex workings of theirs.

We described GFW because they made use of a graphical class-database representation, like UML CD and their article explained the workings of their framework really well . Because of the similarity between GFW and UML CD, and being the inspiration for the article by [SEJ10], their framework was the most useful of the three for a translation to ORM.

Our descriptions of the framework lacked a bit of formal background. This was because in the thesis we focused more on the graphical part, as stated in the introduction. In all of the cases we lacked in formal definitions, we found the formalism not important enough to be included.

Because we only chose to describe three frameworks, we did not address all solutions for problems in PRMs. Still, the frameworks that were not included were not able to give a solution for the biggest problem: representing complex domains. The only article which could have helped us was [SEJ10]. But because this article was already to UML CD specific, it would be hard to map it to ORM. Therefor, GFW was a better option, as it was more general than [SEJ10].

6.2.3 What is Object Role Modeling (ORM)?

The goal of this subquestion was to introduce ORM. We wanted to show why we used ORM for the combination with PRM and what ORM exactly was.

Because we based this subquestion on an article by Terry Halpin, the founder of ORM, we could stay very close to the important basics of ORM. As we were only interested in the graphical part of ORM for this thesis, we did not describe any formalities behind ORM. This

did also limit us a bit in the last subquestion, because we were not able to formally validate our use of P-ORM. Because of the scope of the thesis, this did not matter.

We also wanted to show how the construction of an ORM schema is done. Here, the RE was also of use, as it was the RE we modeled to an ORM schema. Because we could adept the sports domain rather easily, we could also show every important concept that can be used in an ORM schema.

The reasons why we used ORM and not any other modeling language were mostly based on the expressiveness of ORM. A language which comes close is UML with their Class Diagrams. But as this was already done by [SEJ10] for a specific domain and ORM is more expressive than UML-CD, we still found ORM the best choice. Still, UML is better known in the industry than ORM is.

We did not encounter big problems when answering this subquestion. This was mostly because the source for this subquestion, [Hal98], is a very clear and informative article about ORM.

6.2.4 ORM and PRM: a working combination with new possibilities?

As we stated in the introduction, in this subquestion we tried to combine ORM with PRM by mapping a framework from the literature, GFW, to ORM. Also, we wanted to see if ORM could give us more possibilities than GFW gave us.

When converting GFW to ORM, the major problem that we found was the lack of class representation in ORM. Luckily, the PRM that was defined over GFW's relation model was not strictly based on classes, but more on attributes. If this was not the case, the ORM conversion would have been a much more complex. Now, we were able to find a solution for every problem we encountered, except class representation. But as ORM had the *NCN-Advantage*, this was no problem.

Because ORM defines relations in great detail, we thought ORM could give some new possibilities to PRMs, things that in GFW were not possible. Because we did not want to discuss every concept out of ORM, we choose three typical ORM structures. Every extra possibility these structures gave us regarding probabilistic dependencies could be traced back to the way ORM expresses relations. It occurred to us that ORM had the possibility to attach probabilistic not only to the object types itself, but also to the roles they play. In that way, we could isolate certain populations of entity types and attach probabilities to it. Where GFW needed a lot of extra work to do this, ORM already had this possibility in their basic structure. We named this concept the *IWR-Solution*.

We also saw that ORM is just far more expressive and graphical than GFW, giving a PRM a lot of room to express its structure. This was also backed when we look at the constraints and their possibilities in PRM-models. The extra dimension explicit constraints could give P-ORM were very interesting. Because we only discussed two constraints here, options are still open for *uniqueness-*, *subset* and *value constraints*. The downside of this section was that without discussing formalities, we could only hint at the P-ORM usage of constraints. When using constraints in a populated P-ORM schema, it is very likely there are formal problems like illegal populations and strange dependencies.

As with this whole thesis, everything we did was not based on formal validity, but on graphical representations. We did find some interesting constructs, for example connecting

the dependencies to roles. But as we did not formally validate their usage, we do not know for sure if it is really usable in practice. Here lies a good opportunity for future research, as the formal validity of P-ORM still needs to be researched. Also, we did not compare our P-ORM with other modern implementations of PRMs, like the one in [SEJ10]

If we look at the goals we set for this chapter, we discussed a lot about graphical representations and clearly showed ORM could give new possibilities and sometimes even advantages over GFW. ORM already has more expressiveness than most modeling languages, now we also showed this could give them advantages in combination with PRM, P-ORM. Another insight we gained was that ORM models could get very complex when implemented fully. This is also why we did not try to convert our whole RE to P-ORM. What we also did not do was looking at other existing modeling languages, like UML CD, and see if they could do the same as ORM.

6.3 Answering the Research Question

Now we summarized our thesis and evaluated the 'answers' to our subquestions, we can formulate an answer to our main research question. The question was as follows:

How can Probabilistic Relational Models be graphically expressed in ORM and what can be the possibilities of this expression?

Let us start with answering the first part of the question. As Probabilistic Relational Model we used the framework as described by [GFK+07] (GFW) and the existing modeling language used was ORM. As we showed in the fifth chapter, an almost direct translation could be made from the class-representation of GFW to the entity type representation in ORM. We showed that a lot of the concepts in GFW had a counterpart in ORM. The PRM itself, with its probabilistic dependencies could also be mapped almost directly to ORM. The PRM-ORM combination was given the name P-ORM.

The *how* can thus be summarized as: finding counterparts of the PRM in the existing language and in parts express all the important concepts in the existing modeling language. Because ORM is very graphical, it gave us a lot of room to express GFW in it. With some other existing modeling languages this could give more problems.

The main new possibilities P-ORM gives us are based on their way of expressing relations, using roles, constraints and fact types. This enables that combinations of instances of different entity types and sets of instances from one entity type can be used in the probabilistic dependency structure of the PRM. Also, the constraints used in ORM could give new possibilities in the PRM. Examples of this are the usage of constraints as *Gatekeeper* and *Divider*.

We emphasized throughout this thesis that we focused on the graphical representations. Therefor, all the possibilities we found with P-ORM can still give us problems when we are using them in practice. P-ORM still lacks its formal definition and validation.

6.4 Possible Future Work

As we already stated throughout this chapter is that future work on the research of P-ORM is mostly in the formal validation of the framework. We did not address the formalities of certain structures and constraints in ORM and therefor could not address the formalities of those

structures and constraints in P-ORM. Formally defining and validating constructs like roles with probabilistic dependencies, power types with attached dependencies and constraints used in the PRM. Without their formal definition, they only exist in a graphical way. That satisfies the demands for this thesis, but not for usage of P-ORM in practice.

We also fall a bit short in comparing P-ORM to other, newer PRM models, like the framework of [SEJ10]. We do not know if the new possibilities P-ORM could give are also possible in the UML-PRM combination. In [SEJ10], there is no explicit mention of constructs like power types, objectifications etc. As we did no discuss their framework in detail, we can not state that it is not possible at all.

The last point that we want to address as future work is that ORM is very expressive graphically. Many more graphical (and formal) possibilities for probabilistic dependencies in P-ORM can be found. As we did not discuss all constraints, entire populations, schema types and many more extensions there are for ORM, there are many more options for P-ORM. We showed in this thesis that P-ORM *could* work, the next step is to show that P-ORM *will* work.

Bibliography

[BB58] Thomas Bayes and George Barnard. Thomas bayes's essay towards solving a problem in the doctrine of chances. In *Studies in the History of Probability and Statistics: IX.*, volume Vol. 45, No. 3–4 of *Biometrika*, pages 293–315. Biometrika Trust, 1958.

[BFJ03] Olav Bangso, M. Julia Flores, and Finn V. Jensen. Plug & play object oriented bayesian networks. *The Conference of the Spanish Association for Artificial Intelligence. San Sebastian 2003, LNAI 3040*, pages 457–467, 2003.

[BW00a] Olav Bangso and Pierre Henri Wuillemin. Object oriented bayesian networks, a framework for topdown specification of large bayesian networks and repetitive structures. *Technical Report, Aalborg University*, 2000.

[BW00b] Olav Bangso and Pierre Henri Wuillemin. Top-down construction and repetitive structures representation in bayesian networks. In *Proceedings of the thirteenth international Florida Artificial Intelligence Research Society Conference*, pages 282–286. AAAI Press, 2000.

[Coo00] Steve Cook. The uml family: Profiles, prefaces and packages. In Andy Evans, Stuart Kent, and Bran Selic, editors, *UML 2000 -Ů The Unified Modeling Language*, volume 1939 of *Lecture Notes in Computer Science*, pages 255–264. Springer Berlin / Heidelberg, 2000.

[DS96] Debabrata Dey and Sumit Sarkar. A probabilistic relational model and algebra. *ACM Transactions on Database Systems Vol. 21, No. 3.*, pages 339–369, 1996.

[FGKP99] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[GFK+07] Lise Getoor, Nir Friedman, Daphne Koller, Avi Pfeffer, and Ben Taskar. Probabilistic relational models. *Statistical Relational Learning*, 2007.

[GG06] Lise Getoor and John Grant. Prl: A probabilistic relational language. *Mach Learn 62*, pages 7–31, 2006.

[Hal98] Terry Halpin. Object-role modeling (orm/niam). In P. Bernus, K. Mertins, and G. Schmidt, editors, *Handbook on Architectures of Information Systems*, pages 81–105. Springer-Verlag, Berlin, 1998.

[HMK07] David Heckerman, Chris Meek, and Daphne Koller. Probabilistic entity-relationship models, prms, and plate models. *Statistical Relational Learning*, 2007.

[HS09] Catherine Howard and Markus Stumptner. Automated compilation of object-oriented probabilistic relational models. *International Journal of Approximate Reasoning 50*, pages 1369–1398, 2009.

[KP97] Daphne Koller and Avi Pfeffer. Object-oriented bayesian networks. *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97), Providence, Rhode Island, USA*, pages 302–313, 1997.

[LB01] Helge Langseth and Olav Bangso. Parameter learning in object-oriented bayesian networks. *Annals of Mathematics and Artificial Intelligence 32*, pages 221Ű–243, 2001.

[Pfe99] Avrom J. Pfeffer. Probabilistic reasoning for complex systems. *PhD Thesis, Department of Computer Science, Stanford University*, 1999.

[SEJ10] Teodor Sommestad, Mathias Ekstedt, and Pontus Johnson. A probabilistic relational model for security risk analysis. *Computers & Security 29*, pages 659–679, 2010.

[TWG10] Lionel Torti, Pierre-Henri Wuillemin, and Christophe Gonzales. Reinforcing the object-oriented aspect of probabilistic relational models. *Technical Report, UPMC Paris*, 2010.