Fact-Based Specification of a Data Modeling Kernel of the UML Superstructure

Joost Doesburg[1] and Herman Balsters[1]

University of Groningen, The Netherlands

Abstract. Data schemas are an important part of the software design process. The Unified Modeling Language (UML) is the lingua franca in current software engineering practice, and UML class diagrams are used for data modeling within software-engineering projects. Fact-based modeling (FBM) has many advantages over UML, for data modeling. Database engineers that have specified their data schemas in FBM, are often faced with difficulties in communicating these schemas to software engineers using UML. We wish to tackle this communication problem by eventually offering a translation from the FBM-specifications to UML class diagrams. Such a translation requires a formal meta-model description of both FBM and a data-modeling kernel of UML. This paper describes an FBM-based specification of a data-modeling kernel of the UML Superstructure. This kernel will be fact-based, with the added advantage of enabling validation of this FBM-specification.

Keywords: meta-model, fact-based modeling, UML, ORM.

1 Introduction

Fact-based modeling (FBM) is a methodology for modeling the universe of discourse (UoD) of an information system. The main purpose of fact-based modeling is to capture the semantics of the UoD, and to validate results with a domain expert. Unlike object-oriented modeling, FBM treats all facts as relationships. FBM facilitates natural verbalization and thus productive communication (and validation) between stakeholders. FBM is also used as the general name of several fact-based conceptual data modeling dialects, such as Object Role Modeling (ORM), the Natural language Information Analysis Method (NIAM) [6], and Fully-Communication Oriented Information Modeling (FCO-IM) [1]. This paper uses basic ORM2 notations [8], as supported by the NORMA tool [3].

The goals of the Unified Modeling Language (UML) include to provide modelers with a ready-to-use, expressive, and visual modeling language to develop and exchange meaningful methods; support specifications that are independent of particular programming languages and develop processes; and to support higher-level development concepts [14]. In order to support modeling both the static, structural aspects of software and its behavior, UML has multiple diagram types

The authors thank Serge Valera for his contributions.

P. Herrero et al. (Eds.): OTM 2012 Workshops, LNCS 7567, pp. 379–388, 2012.

that support different software aspects [13]. UML is most often used to design object-oriented software, but it is also used to generate conceptual schemas for databases [7,15]. Some authors have compared UML and ORM with respect to data modeling (e.g. [7,11]). This will be treated in some more detail in Sect. 5 of this paper. In this paper, UML version 2.3 is used.

Drawbacks of UML in data modeling have been extensively described in [7,10]. We can safely say that UML was never intended as a conceptual data modeling language. UML does not, for example, have a facility for directly specifying keys in class models, and also it has no facility for directly specifying a large class of constraints specific to roles inside a particular fact type (e.g. most set-comparison constraints, frequency constraints, and ring constraints). UML is used in more than 90 percent of the Fortune 500 companies [14, p. XI]). UML is the lingua franca for modeling in the software development process.

The expressibility of the data-oriented aspects of UML is less than that of FBM [11]. Database engineers that have specified their data schemas in FBM, however, are often faced with difficulties in communicating their FBM-schemas to software engineers using UML. We wish to tackle this communication problem by eventually offering a translation from the FBM-specifications to UML class diagrams. An algorithm for such a translation was created in [4], and is discussed shortly in Sect. 5. There have been other attempts to translate ORM-models to UML-models [2,10]. The algorithm in [10] was only informal, while both algorithms in [2,10] were not implemented. In both cases, the translation was not based on (formal, published) meta-model descriptions of the two languages.

A translation algorithm requires a formal meta-model description of both FBM and a data-modeling kernel of UML. This paper has as its main focus to realize an FBM-based specification of a data-modeling kernel of the UML Superstructure. Since this kernel will be fact-based, we have as an added advantage that this FBM-specification can be validated. It also demonstrates FBM's data-modeling differences with and advantages over UML. As far as we know, our work offers the first fact-oriented specification of a data-modeling kernel of the UML Superstructure.

We note that as a basis for our algorithm, we take that all of the ORM models offered as input to our algorithm are in so-called co-referenced format (CoRef-format). That means that only binary fact types with functional uniqueness constraints have to be considered, along with subtyping arrows and a basic set of constraints. All ORM models can be translated to such a format, as can be found in [10]. Since CoRef-FBM is taken as a starting point, the algorithm can be relatively simple [4]. We note that some advantages of ORM are lost when transforming to Co-Ref form, such as easy, natural verbalization of fact types. We also note that using CoRef-format for our ORM input models will result in a UML-specification that is very close to a relational view of that input model.

The rest of this paper is structured as follows. Section 2 provides a description of our particular choice for a data modeling kernel of the UML Superstructure, based on ORM's Conceptual Schema Design Procedure (CSDP) [10]. Section 3 describes the meta-models of our data-modeling kernel pertaining to pure data

structures, while Sect. 4 provides descriptions (on the meta-level) of the constraint section of the data-modeling kernel. Section 5 offers an overview of the translation algorithm, and offers an example of a translation output. Section 6 summarizes the main contributions, and notes some further research options.

2 The Data Modeling Kernel

As mentioned in the introduction, UML was designed to model the structure and behavior of object-oriented software. For conceptual modeling, not all supported constructs are needed. To determine which UML constructs are required for the data modeling kernel, the Conceptual Schema Design Procedure (CSDP) from [10, pp. 62–63] is used. The CSDP is a structured method to create a conceptual data schema, and is not confined to FBM. The data modeling kernel of UML should contain those constructs that are required by the CSDP.

The first steps of the CSDP are concerned with transforming information examples into elementary facts. UML is object-oriented, so instead of entity-, value-, and fact types; classes, attributes, and associations will be used. These are the basic constructs in UML. Note that these do not directly correspond to the FBM constructs. Some way of defining data types is also necessary, as well as some method of defining derivations.

In the fourth step of the CSDP, uniqueness constraints are added. Some of these constraints can be captured by using UML multiplicity, but this does not cater for all situations. A different approach is presented in Sect. 4.

CSDP step 5 concerns mandatory constraints. While most of the semantics are captured in UML multiplicity, it cannot cater for all situations. Some mandatory constrains will necessarily be captured by comments.

In the next step, value-, set comparison- and subtyping constraints are added. UML enumerations can be used for some value constraints, but UML does not provide graphical support for other value constraints. Subtyping is supported, as well as subtyping constraints. Set comparison constraints can only be captured in specific situations, and are not treated in this paper due to space limitations.

In the last step of the CSDP, other constraints are added, and final checks are performed. We will capture this latter category of constraints by comments.

In the following sections, a formal meta-model for UML for data modeling is defined. The UML superstructure [13] has been followed as closely as possible. Where the text from [13] was unclear or open to interpretation, we have made our own choice for a semantics. These choices will be explicitly mentioned below.

3 Basic Constructs

In this section, the constructs that capture facts from the UoD are introduced.

UML is object-oriented, so facts are captured in classes, attributes and associations. In the UML superstructure [13], every construct is a subtype of ‘element’. While this construct never explicitly occurs in UML diagrams, it provides a reference scheme for all subtypes, and provides the capability to attach comments

to any such element. All elements have a Universally Unique IDentifier (UUID) for machine identification. This is depicted in the FBM-model pertaining to the UML-concept of 'element', as on the left-hand side of Fig. 1.

We deviate from the superstructure in the following manner. Firstly, all abstract subtypes of element were removed, since on a conceptual level, only those constructs that can be instantiated are actually necessary. In the superstructure, an association is also a type. Since we conceive of associations and types as being largely different, we refrain from treating associations as types. Additionally, a comment is an element in the superstructure; since a comment does not have a structural role (it is only a piece of text), we refrain from considering a comment as a subtype of element.

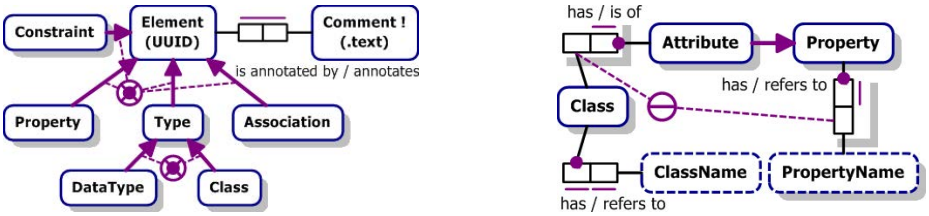

Fig. 1. *Left:* Everything is an element in UML. *Right:* UML class

Classes are the most important data structure for conceptual modeling in UML. A class can participate in associations with other classes, and have any number of attributes. Attribute names are unique within a class. For purposes of user-defined identification, a class also has a unique name. This is depicted in the FBM-model on the right-hand side of Fig. 1.

For conceptual modeling in UML, the methods of a class are not relevant, so these too are not included in the data modeling kernel. Grouping classes into packages is a logical activity, instead of conceptual, so this concept is not included either.[1] The same goes for defining classes as leaf- or abstract classes.

The 'property' construct from the superstructure can have either of two roles in a class diagram. The first role is that of an attribute of a class, and the other is the role of being the end of an association. While the two concepts (i.e. attributes and association ends) share some characteristics (e.g. they both have a multiplicity), their purpose in a class diagram is substantially different. Therefore, in the data modeling kernel, attributes and association ends are defined separately. Each property has a name and multiplicity, and references a type. Properties can be derived, as depicted in the left-hand side of Fig. 2. Note that the 'property' construct as used in UML is different than the one used in mathematics: in UML it is only the supertype of both an association end, and an attribute.

[1] In the UML superstructure, a class name is unique within a package. Since packages are out of scope for this paper, they are considered unique.

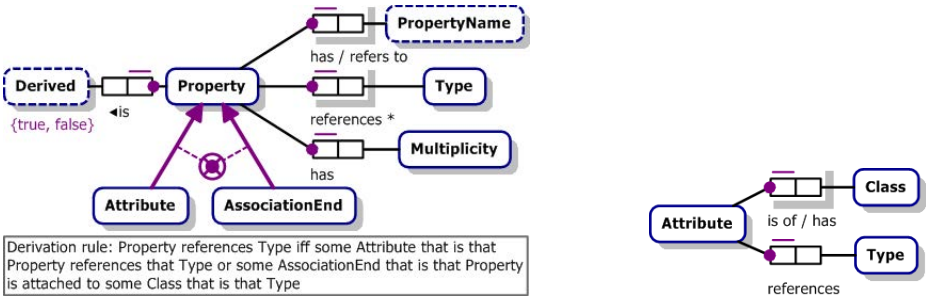

Fig. 2. *Left:* UML property. *Right:* UML attribute

Every attribute is of exactly one class and references one type, as displayed on the right-hand side of Fig. 2. The type of an attribute is generally a data type, but could also be a class.

Figure 3 shows the other function of a property, as an association end. In this setting, the association end gets an aggregation kind, and is attached to a class. An association has two member association ends, can have a (not necessarily unique) name, and can be derived.

In the superstructure, associations of any arity greater than one are allowed. Because CoRef-FBM is taken as a starting point in this paper, we limit ourselves to exactly two association ends per association. The superstructure also allows for the aggregation kind 'shared', but the semantics of this concept are rather vague[2], so we have not included this concept in the data modeling kernel.

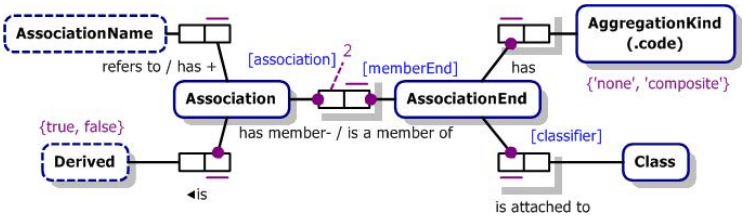

Fig. 3. UML association and association end

UML has a number of primitive data types built in (integer, string, boolean, null and unlimited natural), and provides support for defining new data types. The exact definition of these primitive data types is outside the scope of this paper. As the left of Fig. 4 shows, a data type can also be an enumeration, which we treat in Sect. 4.

[2] "Precise semantics of shared aggregation varies by application area and modeler." [13, p. 38].

Fig. 4. *Left:* UML data type. *Right:* UML multiplicity

4 Constraints

Now that the information-holding constructs have been defined, constraints on that information are investigated.

UML does not use uniqueness and mandatory constraints, but a concept called 'multiplicity'. Multiplicity defines how many instances of a certain property are allowed for a class or association. As the right side of Fig. 4 shows, every multiplicity has a lower bound, and also either has an upper bound, or is unbounded. We always have that lower bound $\leq$ upper bound; an upper bound, if defined, is at least 1.

The superstructure allows for expressions as bounds, but requires that they always evaluate to some integer. This option for expressions has been left out of our model.

The UML construct for subtyping is called a 'generalization', and is depicted in the FBM-model in Fig. 5. A generalization is diagrammatically displayed as a subtyping arrow on a class diagram. Generalizations can only be defined for types, i.e. classes and data types. Every generalization has one type that functions as the supertype, and one that is the subtype. It is possible to constrain a set of two or more generalizations by the use of a 'generalization set', and a generalization can be part of any number of generalization sets. In our version of the kernel, the only constraints that are considered on a generalization set are the 'is covering' and 'is disjoint' constraints. When a generalization set is covering, each instance of the supertype is also an instance of at least one of the subtypes. When a generalization set is disjoint, each instance of the supertype is an instance of at most one of the subtypes. The case where a generalization set is not covering, or not disjoint, is interpreted as the absence of that constraint. Every generalization set has a unique name. In the superstructure, a generalization set constrains zero or more generalizations. As the covering and disjoint constraints are meaningless when the generalization set has less than two generalizations, the two or more constraint was added.

UML provides for a way to define a list of possible (discrete) values: an enumeration. An enumeration has one unique name, and a list of one or more enumeration literals. The names of enumeration literals within an enumeration are unique, as displayed in Fig. 6. As an enumeration is a type, it can be used as the data type for attributes, effectively limiting the values of that attribute to the literals within the enumeration.

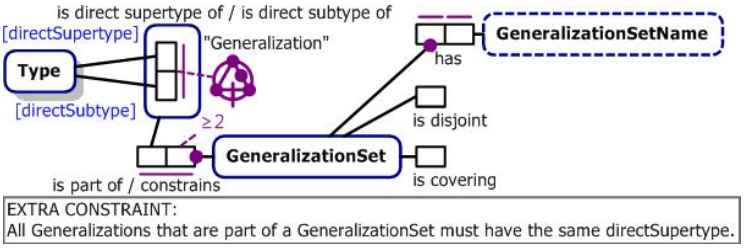

Fig. 5. UML subtyping

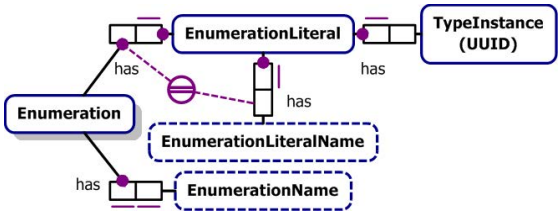

Fig. 6. UML enumeration

The UML superstructure does not support a user-defined reference scheme for objects. While a UUID is sufficient when a computer needs to keep track of objects, it is not suited for human communication. Additionally, the superstructure does not provide a way to define that a combination of properties is unique. In [7], Halpin uses the notations {P} and {Un} to denote primary and secondary identification, respectively. While this is a step up from no value-based identification at all, this is not formally defined in UML yet; also, this approach does not allow for association ends to be used in an identification. Here, an identification mechanism based on [12] is used: UML is extended with an 'identification' construct, as depicted at the top of Fig. 7. This identification construct is added to UML, using the official UML extension method: a profile that defines a stereotype [13]. The UML syntax definition is listed at the bottom of Fig. 7, but keep in mind that the semantics are defined at the top.

The identification construct is analogous to the uniqueness constraint as defined in the FBM exchange schema [5]. Every identification has a unique name, and provides either a primary or secondary identification for a class. An identification is built up from one or more properties (i.e. attributes or association ends). The combination of these properties is unique. Every class in a data schema has one way in which it is preferably identified: through one of its superclasses, or by an identification. Classes can have any number of direct secondary identifications.

For an example of the identification construct in practice, we refer to Sect. 5.

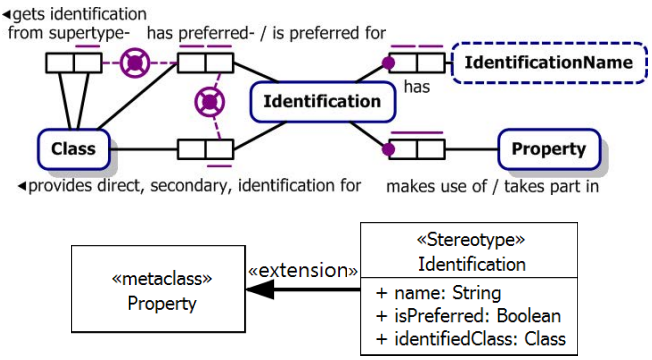

Fig. 7. *Top:* FBM specification of the identification construct. *Bottom:* UML profile for the identification stereotype.

5 Algorithm Results

An algorithm to translate CoRef-FBM schemas into UML was created in [4]. Space limitations do not allow for this algorithm to be thoroughly treated here, but the main decision steps will be presented, along with an example transformation.

Since CoRef-FBM has been taken as a starting point, the algorithm can be relatively simple. Only binary fact types with functional uniqueness constraints have to be considered, along with subtyping arrows and a basic set of constraints.

The algorithm consists of three main steps. First, it is decided which object types translate to a class. Special attention is given to (binary) fact types containing two uniqueness constraints. The second step decides what to do with the fact types, and translates each fact type to an association or an attribute. In the third step, constraints are translated, and identification stereotypes are created.

In the upper part of Fig. 8, an example schema is presented. It features many common modeling constructs: subtyping arrows, internal and external uniqueness constraints, and two value constraints.

In the first step, the algorithm will decide that Person, Teacher and Student will be translated to a class, because they are involved in a subtyping fact type. Booking and CourseEnrollment are also translated to classes, as they have functional roles that do not provide their reference modes. Lastly, SexCode is translated to an enumeration, as it has a list of possible values.

In the second step, all fact types are considered. A fact type, where both object types have been mapped to a class, is translated to an association, while a fact type where only one object type is mapped to a class will be translated to an attribute. The results can be found at the bottom of Fig. 8.[3]

In the last step of the algorithm, all constraints that have not been translated to UML multiplicities are translated. All external uniqueness constraints

[3] The little black triangle on the association designates the reading direction. It does not have any semantics.

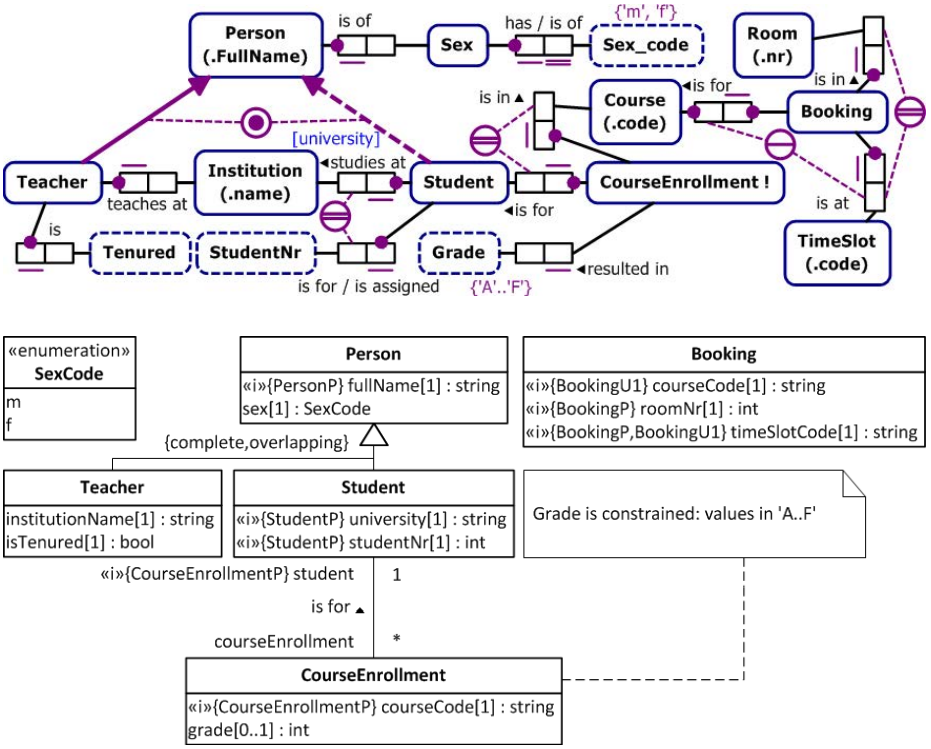

Fig. 8. *Top:* example FBM schema. *Bottom:* the translated UML class diagram

are translated to an identification stereotype. To save space, the name of the stereotype is shortened to «i», the name of the identification includes the name of the identified class, and the name includes a modifier P or U*n*. In the class Booking, the timeSlotCode attribute takes part in two identifications.

The value constraint on the value type Sex_code is translated to the enumeration SexCode. The kernel does not define an equivalent to the value constraint on Grade, so it is translated to a comment.

Note that all fact type readings for attributes are lost, as well as the reverse readings for associations. Because the superstructure does not support many common constraints on data [9], the formality of these constraints is also lost. Note that the resulting UML class diagram is object oriented, so this translation does not help to provide UML with any real fact orientation.

6 Conclusions

Fact-based modeling (FBM) has many advantages over UML, for conceptual modeling. Database engineers that have specified their data schemas in FBM, however, are often faced with difficulties in communicating these schemas to

software engineers using UML. We wish to tackle this communication problem by eventually offering a translation from the FBM-specifications to UML class diagrams. Such a translation requires a formal meta-model description of both FBM and a data-modeling kernel of UML. This paper offers an FBM-based specification of a data-modeling kernel of the UML Superstructure. This kernel is fact-based, with the added advantage of enabling validation of this FBM-specification.

Future research could focus on the formalization of the translation from FBM to UML, and the translation from UML to FBM. It would be interesting to investigate which extra information is needed to perform the latter translation. The required adaptations to the FBM description of the UML meta-model are also of interest.

References

1. Bakema, G., Zwart, J., van der Lek, H.: Fully communication oriented information modelling. Ten Hagen Stam, The Netherlands (2002)
2. Bollen, P.: A formal transformation from object role models to UML class diagrams. In: Proc. of EMMSAD 2002 Workshop, vol. 2 (2002)
3. Curland, M., Halpin, T.: Model driven development with NORMA. In: System Sciences, HICSS 2007, p. 286a. IEEE (2007)
4. Doesburg, J.L.H.: Communicating Conceptual Data Schemas in ESA Space System Projects: Producing UML class diagrams from FBM conceptual schemas. Master's thesis, University of Groningen (April 2012)
5. FBM working group: Fact-based modelling exchange schema. Version 20111021c (2011), http://www.factbasedmodelling.org/
6. Halpin, T.: Object-role modeling (ORM/NIAM). In: Handbook on Architectures of Information Systems, pp. 81–102 (1998)
7. Halpin, T.: UML Data Models From An ORM Perspective: Parts 1–10. Journal of Conceptual Modelling, 1–10 (1998), http://orm.net/uml_orm.html
8. Halpin, T.: ORM 2. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 676–687. Springer, Heidelberg (2005), http://dx.doi.org/10.1007/11575863_87
9. Halpin, T., Bloesch, A.: Data modeling in UML and ORM: a comparison. Journal of Database Management 10(4), 4–13 (1999)
10. Halpin, T.A., Morgan, T.: Information modeling and relational databases, 2nd edn. Morgan Kaufmann (2008)
11. Keet, C.: A formal comparison of conceptual data modeling languages. In: 13th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 2008), Montpellier, France, pp. 16–17. Citeseer (2008)
12. Keet, C.: Enhancing identification mechanisms in uml class diagrams with meaningful keys (2011)
13. OMG: OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.3 (May 2010)
14. Pender, T., McSheffrey, E., Varveris, L.: UML bible. Wiley (2003)
15. Simsion, G.: Data Modeling: Theory and Practice. Technics Publications LLC (2007)