Modeling Z-specification in UML

Snezana Savoska,

University St. Kliment Ohridski - Bitola, Partizanska Str. BB, Bitola 7000, Macedonia
snezana.savoska@uklo.edu.mk

Abstract. Development of software models with specialized notations permits non-important details of the system to be omitted. Z-notation is a tool for specification complex systems using abstract mathematical notation. Further, Z-specifications could be converted for input on software development environments. UML is the most popular notation used today in these environments. The aim of this paper is to investigate this process of conversion of Z-specifications to UML-models. It is based on an example specification of relational model of data.

Keywords: Z-notation, UML, relational data model.

1 Introduction

Z-notation [1] has a long history. In 2002 it was accepted as ISO standard. Z-notation is based on Zermelo–Fraenkel set theory. Z-specifications could be maximally abstracts.

UML [2] has been developed by OMG as notation for object-oriented design. Object-oriented approach with UML design for software development is de-facto industrial standard used in commercially available software development environments.

Modeling in UML a Z-specification is described and discussed in this paper. As a Z-specification is used relational model of data specified in [3].

2 Relational Schema

Every Z-specification begins with some basic data types. In [3] for this purpose are used relational names, relational columns and values:

[*RNAMES*, *CNAMES*, *VALUES*]

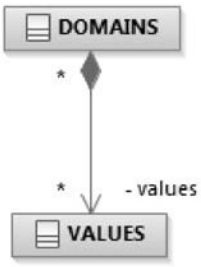
Column names are used in this specification only for extension and they are not modeled in UML. RNAMES and VALUES are modeled initially with classes with the same names and attributes and operations not specified.

Basic types have to be modeled as classes without attributes and operations

sections.

$$DOMAINS == \mathbb{F}_1\ VALUES$$

The type DOMAINS is specified as power set of non-empty finite sets of values (VALUES). In UML, domains are modeled as compositions of values. The empty domain (empty composition) is included. It is not a big deviation from the specification – the model is more general.



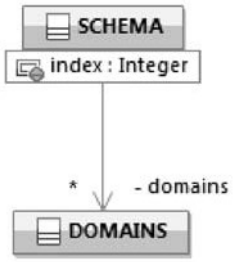
Every value could be used in more than one domain, but could not be component of any domain.

In UML, by default, only finite objects are manipulated. For example, the type Integer is in reality the subset of integers that can be represented on the computer. Here, a composition consists of finite number of elements. In UML, finite characteristic of the artifacts is not specified, because it is by default.

Relational schema is specified as non-empty sequence of domains:

$$SCHEMA == \mathrm{seq}_1\ DOMAINS$$

It is modeled as a class that has qualified association (by attribute 'index') with domains.



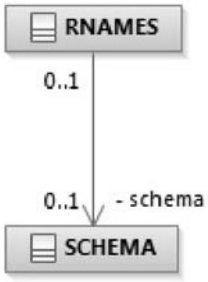
In Z-notation, sequences are indexed with naturals. A sequence in mathematics could be indexed with any countable set. So, it is not a big deviation in that case instead naturals to be used Integer for indexing. The most important property remains: domains in the schema are ordered.

DBSchema

$$db: RNAMES \rightarrowtail SCHEMA$$

Database schema is specified as partial function from relation names to

schemas. Database schema is specified as abstract data type. In UML, the same approach could be used, i.e. database schema could be represented as a class with attributes and operations, where an attribute (or attributes) would map relation names to schemas. This mapping in UML terms is association and it is not recommended association to be hidden with attributes. That the reason why database schema is modeled as association between relation names and schemas.



Multiplicity of this association is zero or one at both ends. The association is directed from relation names to schemas, because navigation in the other direction is not needed at this time. All associations in the diagram are minimal is sense that direction is specified only when navigation in that direction is used.

Initially, the set of relational schemas is empty:

DBSchemaInit

DBSchema

dom $db = \varnothing$

In Z-notation above Z-schema is a constructor. In this case database schema is not modeled with a class and no constructor could be created. The association depends on classes that it connects.

In object-oriented systems, the hypothesis is that, when the system starts, there no objects and only after the initialization new objects and links among them are created. So, by default, the set of links between relation names and schemas, initially, is empty.

DBSAdd

$\Delta DBSchema$

$n?: RNAMES$

$s?: SCHEMA$

$n? \notin \text{dom } db \land$

$db' = db \cup \{n? \mapsto s?\}$

DBSRemove

$\Delta DBSchema$

$n?: RNAMES$

$n? \in \text{dom } db \land$

$db' = \{n?\} \lhd db$

DBSUpdate

$\Delta DBSchema$

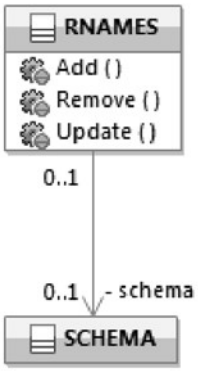
$n?: RNAMES$

$s?: SCHEMA$

$n? \in \text{dom } db \land$

$db' = db \oplus \{n? \mapsto s?\}$

Database schema has three operations: add, remove and update of relation schema. These operations manage links between RNAMES and SCHEMA. The problem now is where these operations have to be located? First location could be the association, i.e. the association between RNAMES and SCHEMA could be with class-association and operations could be placed there. In these case operations have to be static, because they create, remove and update links – instances of the association. The second possibility is to put them on RNAMES, where they would be instance operations with side effect on links of the association. This approach is better, because association, in reality, is implemented with one or two attributes in one or both participating classes (association ends are pseudo-attributes in UML) and every change of the link means change of attributes of these objects. So, this location is used in the UML-model:



There are more possibilities where to place these operations: in class SCHEMA or in another class modeling database schema. These variants go way from original concept. First of them is equivalent to the chosen one only if the association is bidirectional, but it is not true. Second variant could be implemented only with global side effects of the operations that would be hidden in the UML-model.

RNAMES is a basic type in the Z-specification, but in the UML-model it has operations. The effects of last ones are described in OCL pre- and post-conditions. For the operation Add(), pre-condition require no link to exist between the relation name and any schema: `schema->isEmpty()`. Because Add() is now instant operation, it is applied on RNAMES objects and the relation name is the first argument, by default, for Add(). Post-condition of Add() requires a link to be established between relation name and the schema (supplied as argument of Add()) objects: `schema = s`. Post-condition of operation Add() in the UML-model differs from that one in the Z-specification. In UML, post-condition refers only to changed parts and by default all other parts remain unchanged (Frame Problem of UML). In the Z-specification is clearly stated that all other links between relation names and schemas remain the same. Here, the only changed part is the link and that is why in the UML-model of Add() such a post-condition is used. In just a same way, post-conditions of the other two operations are re-mastered.

Relation names in the relational model of data are strings. In the UML-model they are objects. Every object in UML is unique, i.e. it has identity, and it follows that relation names are unique in the UML-model. When the UML-model will be further detailed an attribute of type string will be introduced in RNAMES to represent the symbolic name of the relation. This means that an invariant in RNAMES has to be introduced in the future to warranty that relation names (the string attributes values) are unique. Every RNAMES object has to have a unique name.

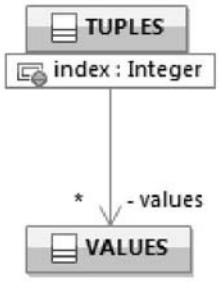
With these operations modeling of database schema is finished. In relational model of data there are logical structure and instance of the database. The first one is the database schema. The second one is a set of all relation instances. In the next section database instance is modeled.

3 Database Instance

Relational instance is a set of all its tuples in the current moment. For this purpose, tuples, first, have to be modeled. Tuple is an ordered set of values.

$$TUPLES == \text{seq}_1\ VALUES$$

Tuples are modeled with qualified association in the same way as schema is modeled with domains. Schemas and tuples can be modeled with attributes. For example, in class TUPLES can be introduced an attribute of type VALUES with '*' multiplicity. This approach hides association between tuples and values and it is not recommended in UML.



Relation is specified with its schema and instance.

RELATION

$DBSchema$

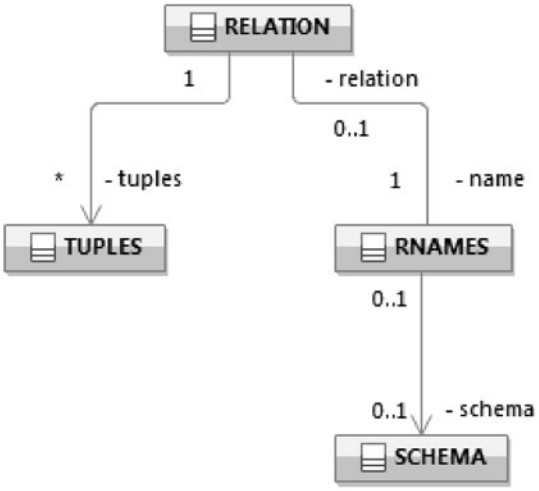
n: $RNAMES$

r: $\mathbb{F}\ TUPLES$

$n \in \text{dom}\ db \wedge$

$(\forall t: TUPLES \bullet t \in r \Rightarrow \#t = \#(db(n)) \wedge (\forall i: 1..\#t \bullet t(i) \in db(n)(i)))$

Relation is modeled in the same way with the class RELATION. This class has association with a relation name. The association shows that every relation has to have relation name and transitively schema, but it is not obligatory every relation name to be bounded with a relation and vice versa. These constraints are specified with the structural notation of UML.



Every relation instance is a set of tuples. It is modeled with an association between relation and tuples. Relation instance could be empty and it is modeled with a star for multiplicity put on the association end at the class TUPLES.

Every tuple participates in exactly one relation. Relation tuples must follow relation schema. Tuples can be associated with relation (relation name) or directly with schema, but closer to the specification is to be associated with a relation (relation object) as it is modeled with two invariants:

```
tuples.values->size() = name.schema->size()
```

and

```
let n:Integer = name.schema->size()
in Set{1..n}->forAll(i | name.schema.domains[i].
values->includesAll(tuples.values[i]))
```

Tuples are lists of values from relational model point of view. In object-relational model every tuple is an object and has its own identity. This means that in object-relational model two tuples can be just same lists of values and to be different objects by their object identifiers. The Z-specification is based on the pure relational model. The UML-model accepts object-relational model: two tuples in one relation can be the same lists of values, but as TUPLES objects to be different tuples.

If pure relational interpretation is needed, in RELATION an additional invariant could be added to warrantee that all tuples in the relation are different only when they are different as lists of values. This invariant could be alternatively part of TUPLES, but that is not the way of the UML-model.

Initially, by the Z-specification, relation instance is empty:

RELATIONInit

$RELATION$

$n?: RNAMES$

$n? \in \text{dom } db \land$

$n = n? \land r = \varnothing$

This initialization of relation instance is a constructor in object-oriented terms. It can be modeled as static operation it RELATION. This operation would bind relation name with relation schema and create an empty instance for that relation. In the UML-model, constructors of all kinds are not modeled to simplify the model.

In the specification supporting Z-schema CHECK is introduced to check for tuple appliance to a relation schema. CHECK is used in relation operations: add and delete tuple. This check doubles relation invariant and is not needed. If Z-specification is extended with checks for errors this supporting Z-schema is needed to separate successful operations from unsuccessful ones, but this is not the case. In UML, there are features specially designed for errors – operation exceptions.

CHECK

$db: RNAMES \pfun SCHEMA$

$n: RNAMES$

$t: TUPLES$

$\#t = \#(db(n)) \land$

$(\forall i: 1..\#t \bullet t(i) \in db(n)(i))$

So, the operation Insert() simply adds and operation Delete() removes a tuple to/from the relation instance, i.e. they add/remove link between the relation and a tuple.

RInsert

$\Delta RELATION$

$t?: TUPLES$

$CHECK[t? / t] \land t? \notin r \land$

$r' = r \cup \{t?\}$

RDelete

$\Delta RELATION$

$t?: TUPLES$

$CHECK[t? / t] \land t? \in r \land$

$r' = r \setminus \{t?\}$

The pre-condition of Insert() requires the new tuple not to be one of the relation instance: `tuples->excludes(t)`, the pre-condition of Delete() requires the opposite: `tuples->includes(t)`.

The post-conditions of Insert() and Delete() operations are:

```
tuples = tuples@pre->union(Set{t})
tuples = tuples@pre - Set {t}
```

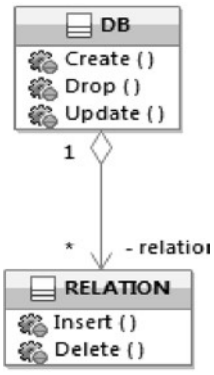
Database instance in the Z-specification is a partial function from relation names into relations

DBInstance

$instance: RNAMES \pfun RELATION$

$\forall n: RNAMES \bullet n \in \text{dom } instance \Rightarrow n = (instance(n)).n$

Here, again the problem is to hide the association with attribute in a class or not. Following UML recommendations, the second approach is used.



Database instance is modeled with the class DB, which is an aggregate of relations. It is possible this aggregate to be empty, but every relation has to be assigned to exactly one database.

Database instance constructor is specified with:

DBInstanceInit

$$DBInstance$$

$$instance = \varnothing$$

This constructor is not modeled following above mentioned reasons.

DBAdd

$$\Delta DBInstance$$
$$n? : RNAMES$$
$$r? : RELATION$$

$$n? \notin \text{dom } instance \land$$
$$n? = r?.n \land$$
$$instance' = instance \cup \{n? \mapsto r?\}$$

DBRemove

$$\Delta DBInstance$$
$$n? : RNAMES$$

$$n? \in \text{dom } instance \land$$
$$instance' = \{n?\} \lhd instance$$

There are supporting operations DBAdd and DBRemove in Z-specification. They add/remove relation instance to/from database instance. The real operations are DBSCreate and DBSDrop.

DBSCreate

$\Delta DBInstance$

$n?: RNAMES$

$n? \notin \text{dom } instance \land$

$(\exists r: RELATION \bullet n? = r.n \land r.r = \varnothing \land$
$instance' = instance \cup \{n? \mapsto r\})$

DBSDrop

DBSRemove

DBRemove

DBSCreate binds a relation name with a schema and create an empty instance for the newly created relation. DBSDrop removes relation schema and its instance. In the UML-model, relational schema and its instance are associated through the class RELATION. This approach is used the relational model – there is no clear notation for relation schema and relation instance as in object-oriented approach for class and class extent. Supporting Z-schemas are not modeled – they are included in modeling of DBSCreate and DBSDrop in the class DB. The last ones are modeled with the operations Create() and Drop(). In UML, it is possible to simplify the operations names, because they are local in the class. In Z-notation, Z-schema names are global and have to be unique. For that reason in Z-specifications a naming convention for Z-schemas has to be used.

Create() pre-condition is: `relation.name->excludes(n)` and its post-condition is: `relation.name->includes(n) and n.relation.tuples->isEmpty()`.

Drop() pre-condition is: `relation.name->includes(n)` and its post-condition is: `relation.name->excludes(n)`.

There is one more operation on database instance – DBUpdate.

$COLUMNS == \text{seq}_1 \, CNAMES$

NSCHEMA

$s: SCHEMA$

$ns: COLUMNS$

$\#ns = \#s$

DBUpdate do not use supporting Z-schemas and it is directly modeled as operation Update() in class DB. Update() pre-condition is: `relation.name->includes(n)` and its post-condition is: `relation->includes(r) and r.name =n`.

Finally, in the Z-specification, there are two more Z-schemas for its extension with named relation columns.

DBUpdate

$\Delta DBInstance$

$n?: RNAMES$

$r?: RELATION$

$n? \in \text{dom } instance \land$

$n? = r?.n \land$

$instance' = instance \oplus \{n? \mapsto r?\}$

This extension is not included in the UML-model, because the model has to re-mastered and step by step modeling would be lost.

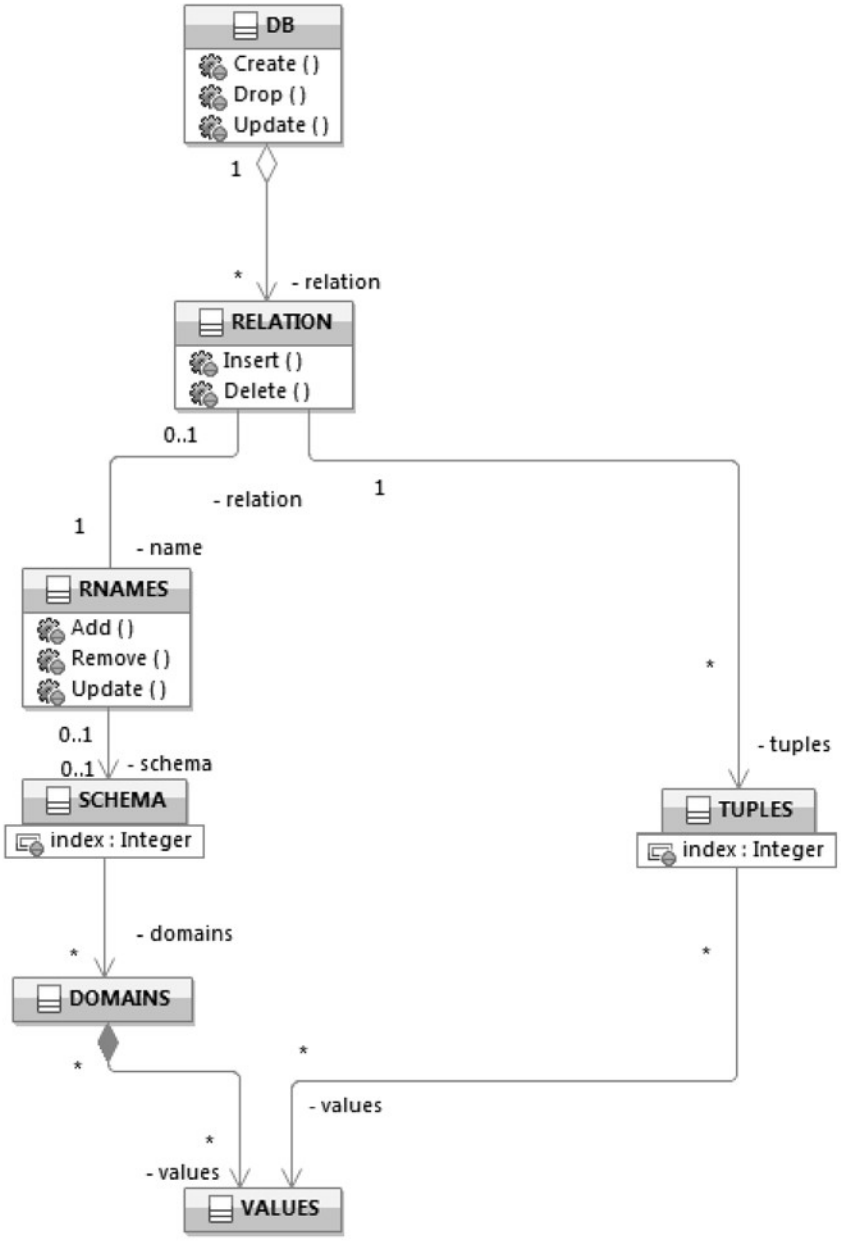
4 Conclusion

The most frequently used approach in UML-modeling of Z-specifications is to unhide hidden associations. Z-notation is a tool for specification of abstract data types. In Z-specification associations are clearly defined with functions. In UML, association may be hidden with attributes and its concept becomes hidden for the reader that is why it is recommended associations to be used instead of attributes.

The UML-model presented here is very abstract – it needs of further re-mastering. For example, the classes RNAMES and SCHEMA form relational database catalog. The last one could be implemented with relations, i.e. the catalog has to be described in terms of relations with self-describing initialization.

The Z-specification is based on the concept for relational model of data described in [4]. There semantics of the model is based on the domains. In the implementations of relational model like DB2, Oracle and so on, this concept is not well supported. These implementations are based on SQL that has been developed as a common query language for relational and hierarchical databases, and as result of that semantics of relational model has been lost.

The whole UML-model as class diagram is:



Relations with column names are called in [4] ‘relationships’. They are specified in [5] and Z-specification there can be used for development of UML model at higher level of abstraction.

Finally, relational model is packed with query language; relational algebra is proposed in [4] as such a language. Z-specification of relational algebra is given

in [6]. The last one is the natural direction for further modeling of relational model in UML.

References

1. ISO/IEC 13568: 2002 (E) Information Technology. Z Formal Specification Notation. Syntax, Type System and Semantics, www.iso.org.
2. Unified Modeling Language, OMG, http://www.uml.org
3. Dimitrov, V.: Formal Specification of Relational Model of Data in Z-Notation, Proc. of XXXIX Conf. of UMB, 178-183 (2010)
4. Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. CACM vol. 18, no 6, 377--387 (1970)
5. Dimitrov, V.: "Relationship" Specified in Z-Notation, Physics of Elementary Particles and Atomic Nuclei, Letters, Vol. 8, No. 4(167), 655—663 (2011)
6. Dimitrov, V.: Formal Specification of Relational Model with Relational Algebra Operations, Proc. of Fifth Int. Conf. ISGT, 25-28 May 2011, Sofia, pp. 57-74.