Model-Driven Engineering for Constraint Database Query Evaluation

María Teresa Gómez-López and Antonia M. Reina-Quintero and Rafael M. Gasca

Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, Sevilla, Spain,
{maytegomez, reinaqu, gasca}@us.es

Abstract. Data used in applications such as CAD, CAM or GIS are complex, but the techniques developed for their treatment and storage are not adapted enough to their needs. Examples of these types of data are spatiotemporal, scientific, economic or industrial information, in which data has not a single value because is defined by parameters, variables, functions, equations .... These complex data cannot be represented nor evaluated with the relational algebra types, then a new, more complex, data type is needed, the *Constraint* type. Constraint Databases were defined and implemented in order to handle this type of *constraint* data. When a Constraint Database is implemented, different configuration parameters can be set up, depending on which database manager is going to be used, which constraint programming tool is going to solve the query evaluation, or which type of constraints can be involved. When some of these parameters are changed, the implementation that supports the evaluation of queries over constraints have to be changed too. For this reason, we propose the use of Model-Driven Engineering to model the queries over Constraint Databases in an independent way of the implementation and the techniques used to evaluate the queries.

1 Introduction

When a great quantity of data is used in a classical application, a database is necessary. However, there are certain types of data that cannot be represented as classical data and treated with relational algebra, such a set of objects located in a zone, where the location of one of the objects may be represented by the constraint $(x + 5)^2 + (y - 2)^2 \leq 36$. This more complex information can be represented using constraints, where a constraint comprises a set of variables, domains and a relation between them. Nevertheless there is no standard database that permits constraints to be handled in the same terms as classical data, and which shields the user from how the information is stored and how the queries are evaluated. In this paper, the term *constraint* is used to describe the data itself, formed by a set of variables in a determined domain, not a restriction over classical attributes. The use of constraints leads to more expressiveness, but it also implies more processes of a complex nature to handle these constraints.

For this reason, some proposals related to Constraint Databases (CDBs) [1], [2], [3] have been developed, since CDBs allow the storage of equations and the variables which relate them as constraints. Hence, the problem is how to store and to evaluate the selection operator, that, depending upon a condition, obtains a horizontal subset of tuples. The selected tuples are those that satisfy a condition, which can relate an attribute either with a constant or with attributes of different relations.

Model-Driven Engineering (MDE) [4] is an approach to software development in which models are used to drive the development of all software artifacts, from code to documentation and tests. MDE is gaining acceptance in several software domains with demonstrated benefits such as cost reduction and quality improvement.

There are a lot of approaches related with Constraint Databases, in which different definitions have been proposed changing the characteristics that describe what a Constraint Database is, and how the queries over them can be evaluated. In this paper, we use the definitions of [5], although the limitation of this solution is that the obtained product is very dependent on the tools and on how the constraints are stored in the database. Thus, if the constraint solver tools, the database manager or the types of constraints are changed, the whole implementation has to be analysed in order to isolate the changes. Then, this paper proposes a set of meta-models related to query evaluation over Constraint Databases which are independent of the types of stored constraints, constraint satisfaction solvers and database management system. The proposal starts from a Platform Independent Model (PIM), where the Constraint Database Metamodel and the queries over the database are defined. We have used MOF [6] to describe the metamodels. In order to evaluate the queries, in this paper we use a metamodel based in Constraint Satisfaction Problems, then we define the Metamodel of it in the Platform Specific Model (PSM), but other models can be defined to use other techniques. The model of a Constraint Satisfaction Problem can be translated into code for an specific Constraint Satisfaction solver, that evaluates the query and reports on which tuples satisfy a condition. OCL cannot be used to describe the constraints, since constraints in Constraints Databases are used as data itself. It means that an evaluation of a query is not the analysis of correctness or satisfiability of the model, that is the common use of OCL. The evaluation of the a query implies the selection of the tuples that satisfy a condition and then will be part of the output relation. Moreover, OCL cannot obtain the possible values of the variables that satisfy a set of constraints, it can only be used to describe the correctness of the instantiation of a metamodel. Note that the evaluation of an OCL constraint is *true* or *false*, while the evaluation of a constraint can be a set of numeric values.

For these reasons, we think that the use of Model-Driven Engineering can be a good way of modeling the system, making independent the definition of the model from a specific implementation. The way to solve the different queries have to take into account a lot of things: the database conceptual model, the related specific constraints, the type of constraints (linear or polynomial), the

conditions of the query, the constraint solver, the domain of the variables... All these characteristics are in different levels of abstraction, but finally all of them are combined to obtain the tuples of the database that satisfy the condition of the query. The main aim of this paper is to define where each of these characteristics have to be located in the levels of modelling proposed in Model-Driven Architecture (MDA) proposed by OMG [7]. MDA solves the problem of interoperability by using conceptualized (formal) system models. The key idea is the successful integration and interoperability in the intelligent use and management of metadata across all applications, platforms, tools, and databases.

In the Platform Independent Model (PIM), we propose to describe the metamodels related to Constraint Databases, Constraints and Queries over Constraint data. The details of the Platform Specific Model are related to the use of Constraint Satisfaction Problems to evaluate the queries. Then, the PSM can be transformed into Code that could be solved by means of Choco^{TM} [8], Jsolver^{TM} [9], Comet^{TM} [10]... In general, the needed metamodels are:

- The metamodel of Numeric Constraints.
- The metamodel of Constraint Databases.
- The metamodel of selection query over the Constraint Database metamodel.
- The metamodel of Constraint Satisfaction Problem.
- The metamodel of Constraint Databases combined with Constraint Satisfaction Problem metamodel.

This work is organised as follows: Section 2 presents an example of Constraint Database, and what types of queries can be evaluated. Section 3 shows the MDA-based modelling architecture proposed in this paper. The metamodels defined in each modelling level are detailed in the following sections. Section 4 presents the needed definitions and an example about Constraint Database and Constraint to understand the proposed metamodel. Section 5 shows the metamodel of the selection operator for Constraints stored in a Constraint Database. Section 6 describes the metamodels defined in the PSM level, Constraint Satisfaction Problems and Constraint Databases for query evaluation. Section 7 analyses the previous works related to this paper. Finally, conclusions and future work are presented.

2 Motivating Example

An example of query evaluation over Constraint Data can be the information presented in Figure 1. Figure 1 shows a map in which cicles represent the broadcasting areas of differente radio stations, and Figure 2 shows how this information can be stored in a Constraint Database by means of two tables. One of them stores information about cities and the other one about broadcasting areas. The tables *Cities* and *Radio Stations* are formed by classical attributes (such as idCity or Name) and Constraints attributes (such as Location). The queries over the tables can define conditions over constraint or classical attributes, being necessary a solver to know the tuples of the relations that will be part of the output

relation of the queries. An example of query over these tables can be to obtain the frequency of the radio stations available in all places of a city:

$$\prod_{radioStations.Frequency}(\sigma_{cities.location \subseteq RadioStations.location}(cities \bowtie radioStations))$$

This paper focus on the selection operator (σ) over constraint data stored in a Constraint Database, and how these queries can be evaluated.

Fig. 1. Broadcast of radio stations

3 MDA Architecture for Constraint Database Query Evaluation

In this section we describe the MDA-architecture to model the possible queries over Constraint Databases. This architecture is depicted in Figure 3, where the three levels of modeling are shown, although this paper is focused on the PIM and PSM levels since its aim is to specify the metamodels that are needed in these levels. The main metamodel in the PIM level is the Constraint Database Query metamodel which uses the Constraint Database and Constraint metamodels. The evaluation of the queries is modeled with the CDB-CSP metamodel, that uses the CSP metamodel. Note that models that conform the CDB-CSP metamodel are obtained by means of a model-to-model transformation. This transformation has as input a model that conforms the Constraint Database Query metamodel. The notation used in Figure 3 is based on that used by Gronback [11]. Finally

Cities

IdCity	Name	Location
...	...	...
41	Seville	y <= 219+0.5x AND y <= 506-0.4x AND y <= 1267-7x+0.01x*x AND y >= 753-2x AND y <= 9.25311x-2071
...	...	...

Radio Stations

IdRadioStation	Frequency	Location
101	94.1	(x-190)*(x-190)+(y-267)(y-267)<=170*170
102	97.8	(x-370)*(x-370)+(y-131)(y-131)<=66*66
103	101.2	(x-423)*(x-423)+(y-307)(y-307)<=73*73
...	...	...

Fig. 2. Example of tables with Constraint Data

note that the code for different solvers (Choco, JSolver and Comet) can be obtained from the CDB-CSP metamodel by means of different model-to-text transformations.

4 Platform Independent Model for Constraint Database Development

Many database applications have to deal with an enormous quantity of data which can even be infinite such as data for time and space. However, databases have a finite capacity. Constraints represent the relation between different variables, and can therefore represent an infinite group of values of variables in a compact way, by using a single appropriate expression.

The basic idea behind the CDB definition is to generalize the notion of a tuple in a relational database to a conjunction of constraints, since a tuple in relational algebra can be represented as an equality constraint between an attribute of the database and a constant.

4.1 CDB Definitions and MetaModel

The definition about Constraint Databases used in this paper is based on Chapter 2 of [12] with some variants proposed in [5] where the entities in a CDB can be formed by classical attributes (an equality between a variable and a constant), and constraint attributes (set of variables defined over a domain that describe the possible correct values that the variables can take).

First of all, it is necessary to introduce what a constraint is. A Constraint is a relation between a set of variables defined over a domain [13], in which there is a limitation over the values for the instantiation of the variables.

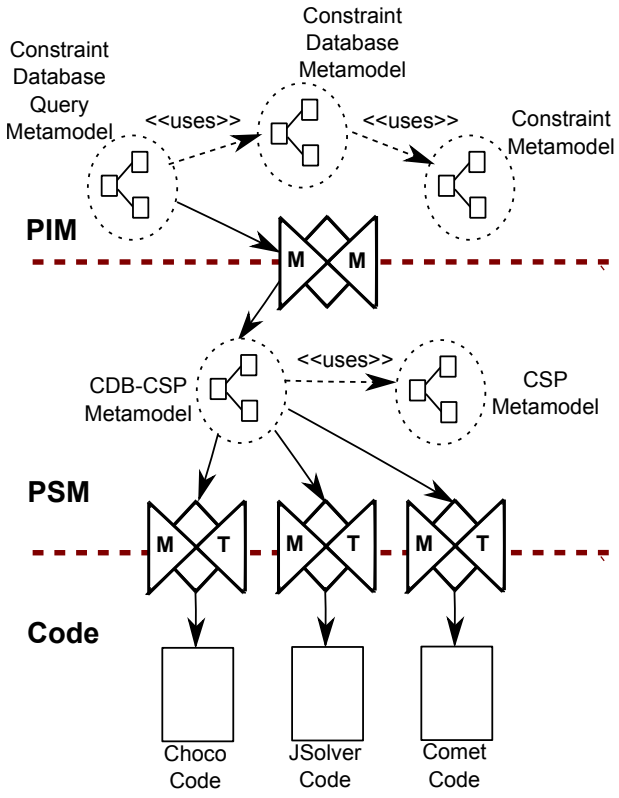

Fig. 3. MDA-based architecture

Definition 4.1: Constraint. Let Ω be a vocabulary, thereby a constraint over Ω is a first-order formula over Ω, and is also called an Ω-constraint.

The previous definition can be expressed with the following grammar:

```
Constraint := Atomic_Constraint BOOL_OP Constraint
  | '¬' Constraint ;
  | Atomic_Constraint ;
BOOL_OP:= '∨' | '∧' ;
Atomic_Constraint:= function COMPARATOR function ;
function:= Var FUNCTION_SYMBOL function
  | Var
  | Value ;
COMPARATOR:= '=' | '<' | '≤' | '>' | '≥' ;
FUNCTION_SYMBOL:= '+' | '−' | '*' | '/' ;
```

This grammar can be described with the Constraint metamodel shown in Figure 4. A *Constraint* is represented as an *Atomic Constraint*, a negation of a

constraint ($\neg$), or a binary combination using logic operations ($\vee$, $\wedge$) of *Constraints*. An *Atomic Constraint* is the comparison of two *Functions* using comparators operators ($=$, $<$, $>$, $\leq$, $\geq$). Each *Function* is a combination using arithmetical operations of numeric variables, instantiated and non-instantiates, where each *Variable* is defined over a domain.

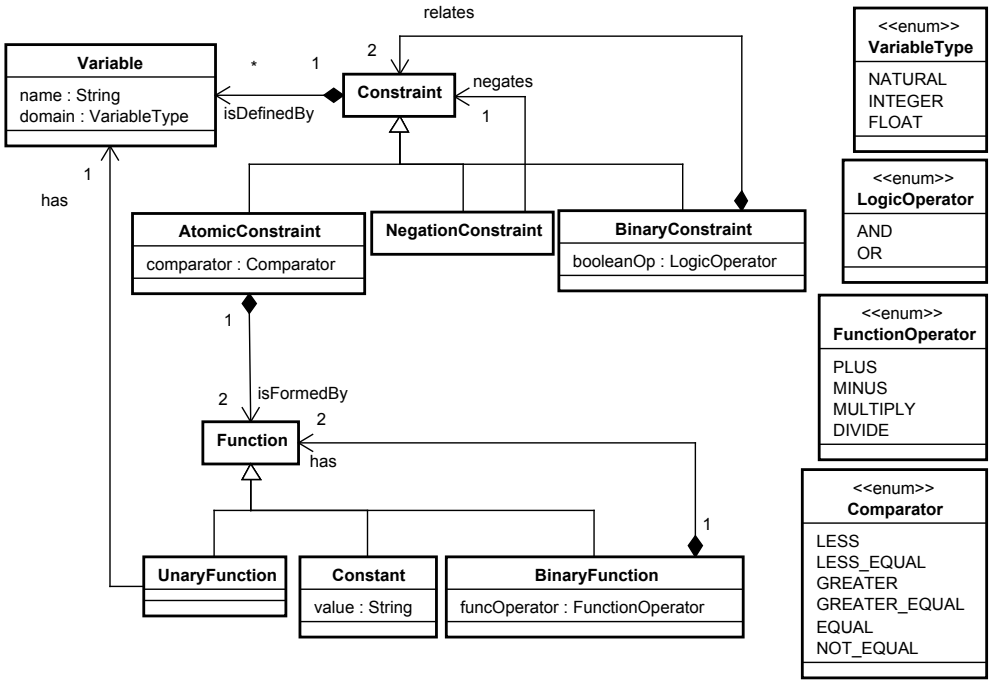

Fig. 4. Constraint metamodel

- A *constraint k-tuple* with the variables $x_1, \ldots, x_k$ over the vocabulary Ω is a finite conjunction $\varphi_1 \wedge \ldots \wedge \varphi_N$ where each φ_i, for $1 \leq i \leq N$, is either a constraint such that $\{x_j{=}Constant\}$, where $x_j \in \{x_1, \ldots, x_k\}$, called *Classical Attribute*, or an Ω-constraint over the variables $x_1, \ldots, x_k$ which do not correspond to a classical attribute, called *constraint attribute*.
- A *constraint relation* is defined as a finite set of *Classical Attributes* and *Constraint Attributes*. A *constraint relation of arity k*, is a finite set $r = \{\psi_1, \ldots, \psi_M\}$, where each ψ_j for $1 \leq j \leq M$ is a constraint k-tuple over $\{x_1, \ldots, x_k\}$. The corresponding formula is the disjunction $\psi_1 \vee \ldots \vee \psi_M$, such that $\psi_j = \varphi_1 \wedge \ldots \wedge \varphi_N$ for each φ_i is a *constraint k-tuple*, where $1 \leq i \leq N$. If in each $\psi_j \in r$ there is a φ_i such that $\{x{=}Constant\}$, where x is the same variable in all φ_i belonging to different ψ_j, and x does not

appear in the rest of the φ_i of the same ψ_j, then the x variable is a **classical attribute**, while the rest of the variables belong to **constraint attributes**.

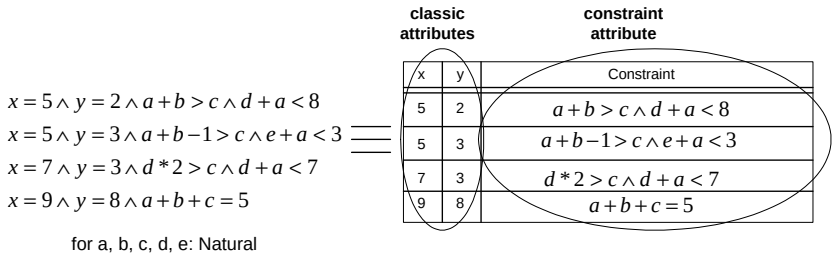

Fig. 5. Constraint k-tuples and constraint relation example

A relation has classical attributes if and only if:

φ_{ij} is a $\varphi_i \in \varphi_1 \wedge \ldots \wedge \varphi_N$ and $\psi_j \in \psi_1 \vee \ldots \vee \psi_M$, such that $\psi_j = \varphi_{1j} \wedge \ldots \wedge \varphi_{Nj}$, then a constraint relation will have a classical attribute (x) if:

$\exists \varphi_{ij} \bullet \forall j \in 1..M \mid i \in 1..N, \{\varphi_{ij} \equiv x = c_j\}$
$\wedge \forall t \in 1..N \wedge t \neq i \wedge \varphi_{tj}(x_1, \ldots, x_k) \wedge x \notin \{x_1, \ldots, x_k\}$

where c_j is a constant, M the number of tuples and N the number of attributes (columns).

This implies that if an equality relation exists between a variable and a constant (the same variable in all tuples) in all constraint k-tuples, and that if this variable does not appear in another constraint attribute, then this variable is a classical attribute, since this variable follows the relational algebra.

An example is presented in Figure 5, where the shown relation (table) is composed of one constraint attribute and two classical attributes. These data can be represented exclusively with constraints, as it is shown in the left side of the picture, or using a table with different types of attributes.

– Therefore, a *Constraint Database* is a finite collection of constraint relations composed of **Classical and Constraint Attributes**. Derived from the combination of classical an constraint attribute, the metamodel that describes the Constraint Database is shown in Figure 6. This metamodel represents that a CDB is described by a set of tables which attributes can be Constraint or Classical (String, date, numeric, ...). In the same way, each table is formed by different tuples whose values depend on the types of the attributes of the relation.

5 Modeling Constraint Database Queries

The relational data model proposed by E.F. Codd (1970) has all the information structured logically using relations (tables). Each relation has a name and is

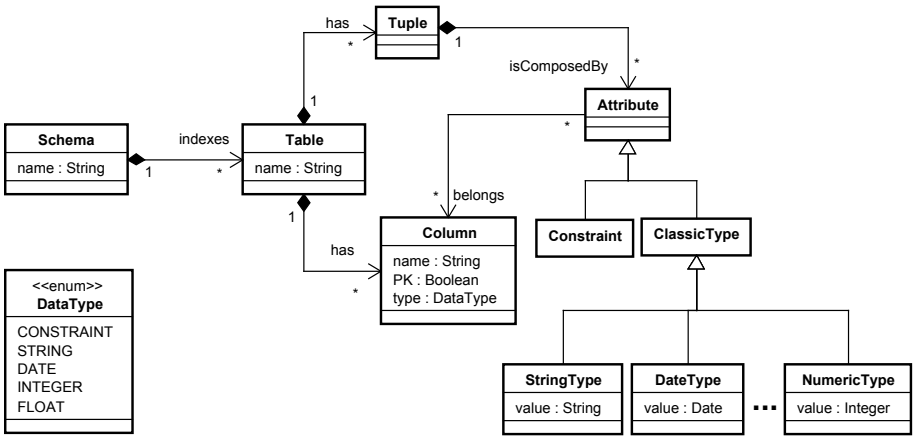

Fig. 6. Constraint Database metamodel

formed by attributes (columns), and where each tuple (row) contains a value for each attribute. Therefore, a relational database is a collection of normalised relations where each relation has a different name. The schema of CDB is very similar, but with the difference that attributes, can be constraint or classical attributes. One of the main problems in CDB is that the algebraic operators have to be syntactically and semantically extended, and it is also necessary to evaluate them over constraint attributes. In this paper, only the selection operator is analysed, being the rest of operators the objective of a future work.

5.1 Selection Operator for CDBs

Selection Operator (σ) has the signature $R_1 = \sigma_{predicate}(R_2)$, where R_1 and R_2 are relations and *predicate* is a condition involving attributes of relation R_2. Relation R_1 is a subset of relation R_2, and represents those tuples which satisfy the conditions of the predicate. A predicate is a Boolean expression whose operators are the logical connectives $\chi = \{and, or\}$ and arithmetic comparators $\theta = (<, \leq, >, \geq, =, \neq)$, and whose operands are either attribute names or domain constants. This means that the predicate is represented as:

$a_1 \theta c_1 \ \chi \ a_2 \theta c_2 \ \chi \ \ldots \ \chi \ a_n \theta c_n$

where a_i is an attribute and c_i can be a constant of the domain of the attribute a_i or another attribute of the same domain than a_i.

The main difference of the CDB selection operator with respect to classical relational algebra is when the attributes are constraints. Hence, the redefinition of the comparators ($<, \leq, >, \geq, =, \neq$) is necessary, and the possible extension of new comparators such as ($\subset, \subseteq, \supset, \supseteq, \&$).

In the selection operator each tuple of the input relation is analised in order to obtain a new relation only with the satisfiable tuples. In the next section, we explain the meaning of the comparators for constraints attributes.

5.2 Types of comparators between constraints attributes

Let C_x and C_y be two constraints where $X = \{x_1, x_2, \ldots, x_n\}$ are the variables of C_x, and $Y = \{y_1, y_2, \ldots, y_m\}$ are the variables of C_y. If $x_i \in X$, and $y_j \in Y$, then $x_i \equiv_{\mathcal{S}} y_j$ is true if both variables are syntactically equal, it means that they have the same name.

The comparison operators between constraints are:

- $C_x < C_y$ is true if for all $x_i \equiv_{\mathcal{S}} y_j$, the maximum value of x_i is smaller than the minimum value of y_j. This definition can be extended to $C_x > C_y$. An example based of Figure 7 can be that $A < D$ but $\neg(C < D)$.

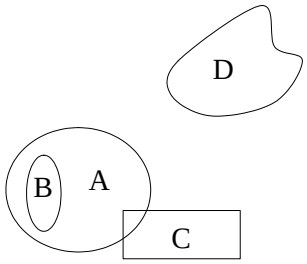

Fig. 7. Example Of Relation between Constraints

- $C_x \leq C_y$ is true if for all $x_i \equiv_{\mathcal{S}} y_j$, the maximum value of x_i is smaller or equal than the minimum value of y_j. This definition can be extended to $C_x \geq C_y$. An example based of Figure 7 can be that $B \leq D$ but $\neg C \leq D$.
- $C_x = C_y$. In this case is not possible compare both constraints syntactically, due to two constraints can be morphologically different but to have the same solutions. $C_x = C_y$ is true if all the solutions of C_x for $x_i \equiv_{\mathcal{S}} y_j$ are solutions of C_y and vice versa. It means that there are no solutions of C_x that do not to belong to C_y. In this case, it is not necessary to rename the variables, due to C_x and C_y could share solutions. The operation <>, which describes the inequality of two constraints, represents if two constraints are different. For the example shown in Figure 7, there are not two constraints equals, but for example it is possible to ensure that $A <> B$.
- C_a & C_b is true if there is a solution that satisfied both constraints, where all the variables are syntactically equal. An example based of Figure 7 can be that $C \& A$ but $\neg(C \leq B)$.
- $C_a \subseteq C_b$ is true if all the solutions of C_a are also solutions of C_b. In order to analyse the inclusion operator in constraints, both constraints have to be defined over the same variables. Therefore, since C_a and C_b are two constraints where $X = \{x_1, x_2, \ldots, x_n\}$ are the variables of C_a and C_b, then $C_a \subseteq C_b$ is equal to the implication $(C_a \rightarrow C_b)$ [14]. This comparison determines if all the solutions of C_a are also solutions of C_b, although it is

possible that C_b has solutions that do not belong to C_a. An example based of Figure 7 can be that $B \subseteq A$ but $\neg(C \subseteq A)$. This definition can be extended to $C_x \supseteq C_y$.

– $C_a \subset C_b$ returns *true* if all the solutions of C_a are also solutions of C_b, and C_b has at least one solution which does not belong to C_a. As for the previous operator, both constraints have to be defined over the same variables. This definition can be extended to $C_x \supset C_y$.

5.3 Constraint Query Extension Metamodel

Figure 8 presents the metamodel of a simplification of the constraint query language explained in the above section. This metamodel represents that a query is defined as the evaluation of a condition over two attributes of the involved tables, although it is a simplification of the problem since in the relational algebra a boolean combination of conditions are allowed. The type of comparators ($<$, $\leq$, $\subset$, $\subseteq$, ...) depends on the types of attributes involved in the query (Constraint, String, Date, ...).

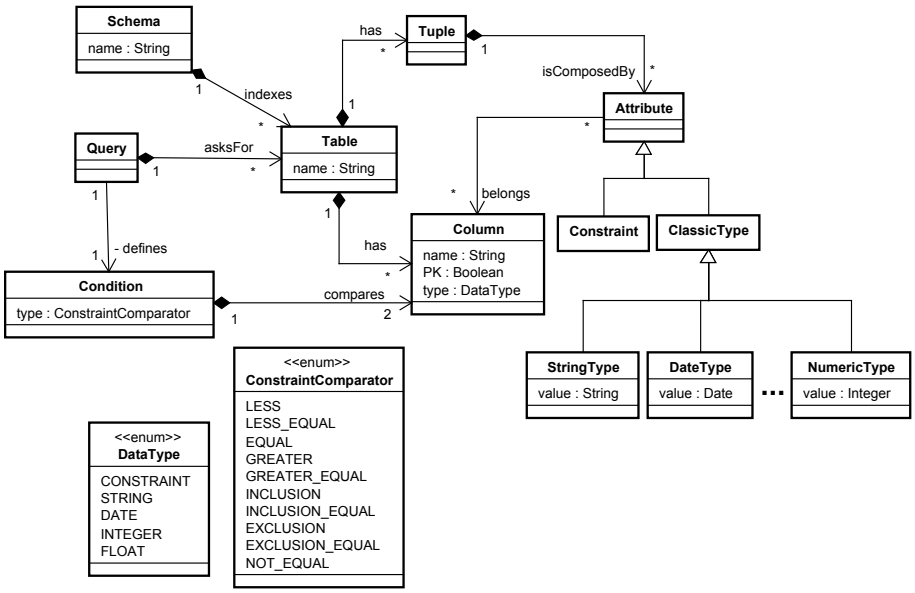

Fig. 8. Constraint Query Metamodel

6 Platform Specific Model for Constraint Query Evaluation

The metamodels explained in the previous sections can be transformed into other metamodels according to different techniques, for example neural networks or symbolic elimination paradigm. In this paper we explain how Constraint Satisfaction Problem metamodel can help us to model solutions that can be transformed into an specific code in function of the solver tool. First, we need to explain what is a Constraint Satisfaction Problem.

6.1 Constraint Satisfaction Problem Metamodel

In order to work with constraints, constraint satisfaction problems are built dynamically to obtain the solution of each query over ORCDBs. Hence, some aspects about constraint satisfaction problems are presented.

Problems of engineering have been modelled as Constraint Satisfaction Problems (CSP) [15] and [16]. A Constraint Satisfaction Problem is a representation and reasoning framework consisting on variables, domains, and constraints. Formally, it is defined as a triple $<\mathcal{X}, \mathcal{D}, \mathcal{C}>$ where $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables, $\mathcal{D} = \{d(x_1), d(x_2), \ldots, d(x_n)\}$ is a set of domains of the values of the variables, and $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ is a set of constraints. Each constraint $\mathcal{C}_i$ is defined as a relation $\mathcal{R}$ on a subset of variables $\mathcal{V} = \{x_i, x_j, \ldots, x_k\}$, called the *constraint scope*. The relation $\mathcal{R}$ may be represented as a subset of the Cartesian Product $d(x_i) \times d(x_j) \times \ldots \times d(x_k)$. A constraint $\mathcal{C}_i = (\mathcal{V}_i, \mathcal{R}_i)$ specifies the possible values of the variables in $\mathcal{V}$ simultaneously in order to satisfy $\mathcal{R}$. Let $\mathcal{V}_k = \{x_{k_1}, x_{k_2}, \ldots, x_{k_l}\}$ be a subset of X. An l-tuple $(x_{k_1}, x_{k_2}, \ldots, x_{k_l})$ from $d(x_{k_1}), d(x_{k_2}), \ldots, d(x_{k_l})$ is called an *instantiation* of variables in $\mathcal{V}_k$. An instantiation of all variables in $\mathcal{X}$ is a solution.

A CSP can be solved by searching using standard backtracking algorithms [17], [18]. Usually, a combination of search with consistency techniques is used to solve these problems. The consistency techniques remove inconsistent values from variables' domains during the search. Several local consistency and optimization techniques have been proposed as ways of improving the efficiency of search algorithms.

The metamodel that we propose is shown in Figure 9, that represents a CSP as a set of constraints formed by variables defined for a domain. Each variable is described by a domain and the minimum of maximum values that can take.

6.2 Metamodel of Constraint Databases Query evaluation using CSPs

In order to represent the condition that each tuple has to satisfy to belong to the resulted relation, we propose the use of CSPs. The metamodel is very similar to the Constraint Database Metamodel, since relational algebra has defined the closure property, then the inputs and outputs of a query are relations. The main

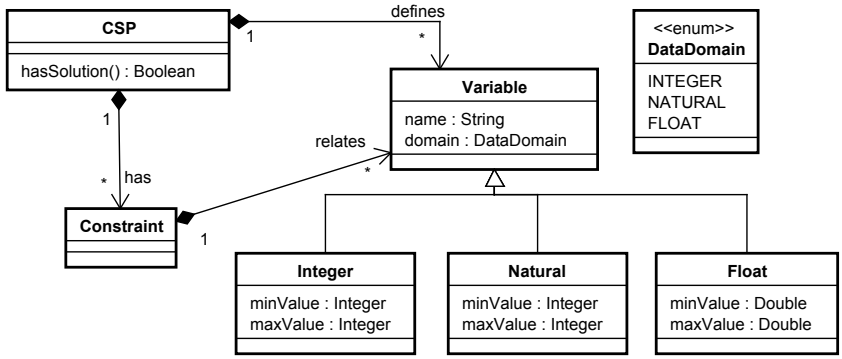

Fig. 9. Constraint Satisfaction Problem Metamodel

difference is that there is a BooleanCSP class associated to each tuple. This class is also related to a CSP model. The BooleanCSP class represents if to find a solution of the CSP implies that this tuple will form part of the output or not. For example, to know if $C_a \subseteq C_b$, a CSP model has to be built to determine if exists a satisfiable value of C_a where C_b is not satisfiable. If this value is found, we can ensure that $C_a \subseteq C_b$ is fault. For this example the value of the findASolution attribute of the class BooleanCSP will be false in this example.

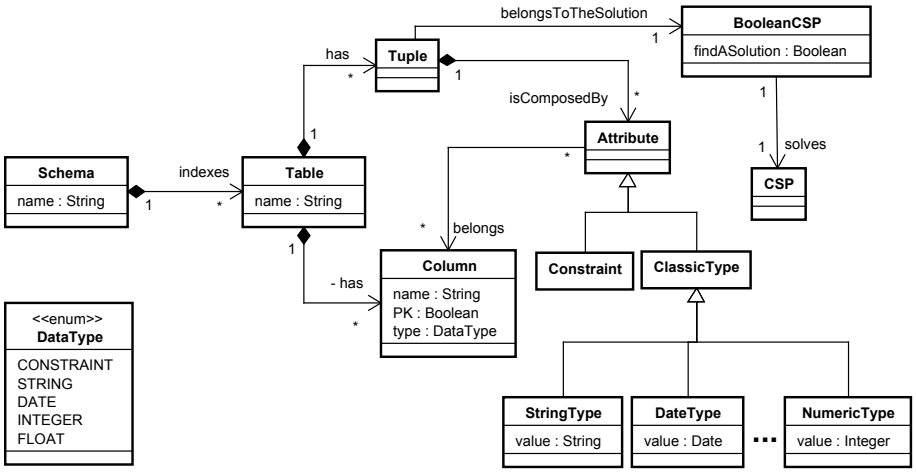

Fig. 10. Constraint Database with CSP Metamodel

7 Related Work

CDBs were defined in 1990 with a paper by Kanellakis, Kuper and Revesz [1]. The first works about query language were based on a subset of Prolog (Datalog [19]) and Constraint Logic Programming (CLP) [20], that were used to define Constraint Databases [2], [3]. Different solutions have been developed, for example: DISCO [21] based on Datalog; MLPQ/PReSTO [22] for Spatio Temporal Object; DEDALE [23] uses the object-oriented paradigm; CCUBE [24] is an integration of constraint calculus for extensible constraint domains within monoid comprehension; CQA/CDB [25] that uses linear constraints for spatiotemporal area. The main disadvantages of the proposals mentioned above are that none of them offers a versatile solution for any type of application, neither do they offer a general solution independent of the nature of the problem domain or the platform. Most of these prototypes are developed to work with spatiotemporal data such as those in [26][27], since the domain is specific. Another important issue is that some of the described query languages have a syntax greatly differ from SQL. It implies that although they use similar models to represent Constraint Databases and the query operators, each of them has a totally different solution to store, represent and evaluate the queries.

In other works, Constraint Satisfaction Problem has been used in Model-Driven Engineering to transform models with OCL constraints [28] into models evaluated for Constraint Satisfaction Solvers. In [29], an extension of QVT is proposed to avoid the transformation by hand of a model with OCL rules into a model that can be evaluated in terms of constraints. Also in [30] is analysed how the use of Constraint Satisfaction Problem can be applied to detect inconsistences in the models.

In [5] the proposed solution is based on how to store the constraints in an Object-Relational Database [31], that could be treated from the point of view of Model-Driven Engineering as is proposed in [32]. How Model-Driven Engineering can help in the Object-Relational Databases development has been analysed in [32], the problem of this representation is the query evaluation of the constraint data. The idea of our paper is related to a way of representing Constraint data when they are stored in an Object-Relational Database, and evaluate queries over data. Likewise, we should consider the works from Atzeni, Bernstein et al. on model management [33].

However, to the best of our knowledge there are no previous works applying model-driven techniques for Constraint Satisfaction Problem in Query evaluation development.

8 Conclusions and Future work

In this work we propose the use of MDE in the definition of Constraint Databases and the evaluation of the selection operator. For this, three different metamodels have been defined in the PIM Level (Constraint, Constraint Database and Constraint Database Query). Also the PSM level has been discussed, since we

propose the use of CSPs to evaluate the selection operator over the constraint stored in a CDB. In the PSM level two metamodels have been defined (Constraint Satisfaction Problem and CDB with CSPs).

As future work, we propose to enrich the query metamodel for boolean combination of conditions. Also we consider interesting to extend the query evaluation to the rest of primitive operators of relational algebra: projection, union, cartesian product and difference.

Acknowledgment

This work has been partially funded by the Junta de Andalucía by means of la Consejería de Innovación, Ciencia y Empresa (P08-TIC-04095) and by the Ministry of Science and Technology of Spain (TIN2009-13714 and TIN2010-21744-C02-01) and the European Regional Development Fund (ERDF/FEDER).

References

1. G. M. Kuper P. C. Kanellakis and P. Z. Revesz. Constraint query languages. *Symposium on Principles of Database Systems*, pages 299–313, 1990.
2. Peter Z. Revesz. Datalog queries of set constraint databases. In *ICDT'95: Proceedings of the 5th International Conference on Database Theory*, pages 425–438, London, UK, 1995. Springer-Verlag.
3. Peter Z. Revesz. Safe query languages for constraint databases. *ACM Trans. Database Syst.*, 23(1):58–99, 1998.
4. Douglas C. Schmidt. Model-driven engineering. *IEEE Computer*, 39(2):25–31, February 2006.
5. María Teresa Gómez López, Rafael Ceballos, Rafael M. Gasca, and Carmelo Del Valle. Developing a labelled object-relational constraint database architecture for the projection operator. *Data Knowl. Eng.*, 68(1):146–172, 2009.
6. OMG. Meta object facility 2.0 core final adopted specification. 2003.
7. OMG. Mmda guide version 1.0.1. 2003.
8. Reference Manual. Choco solver. 2011.
9. Reference Manual. Ilog jsolver. 2001.
10. The Comet Hybrid Optimization Platform 2.1.1. Dynadec. 2010.
11. Richard C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.* Addison-Wesley Professional, 2009.
12. P. Revesz. *Spatial Constraint Database.* Springer, 2001.
13. Rina Dechter. *Constraint Processing (The Morgan Kaufmann Series in Artificial Intelligence).* Morgan Kaufmann, May 2003.
14. Kim Marriott and Peter J. Stuckey. *"Programming with Constraints. An introduction", Simplification, Optimization and Implication.* The Mitt Press, 1998.
15. Martin Sachenbacher and Brian Williams. Diagnosis as semiring-based constraint optimization. In *15th International Workshop on Principles of Diagnosis*, pages 15–20, 2004.
16. Albert Croker and Vasant Dhar. A knowledge representation for constraint satisfaction problems. *IEEE Trans. Knowl. Data Eng.*, 5(5):740–752, 1993.
17. J.R. Bitner and E. M. Reingold. Backtracking programming techniques. *Comm of the ACM*, 18(11):651–656, 1975.
18. Solomon W. Golomb and Leonard D. Baumert. Backtrack programming. *J. ACM*, 12(4):516–524, 1965.

19. S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, 1989.
20. Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *J. Log. Program.*, 19/20:503–581, 1994.
21. Jo-Hag Byon and Peter Z. Revesz. Disco: A constraint database system with sets. In *CDB*, pages 68–83, 1995.
22. P. Z. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu, and Y. Wang. The mlpq/gis constraint database system. In *ACM SIGMOD Conference, May 16-18, Dallas, Texas, USA*, page 601. ACM, 2000.
23. S. Grumbach, P. Rigaux, and L. Segoufin. The DEDALE system for complex spatial queries. In *SIGMOD Conference*, pages 213–224, 1998.
24. A. Brodsky, V. E. Segal, J. Chen, and P. A. Exarkhopoulo. The CCUBE constraint object-oriented database system. In *ACM SIGMOD Conference*, pages 577–579. ACM Press, 1999.
25. D. Goldin, A. Kutlu, and M. Song. Extending the constraint database framework. In *PCK50*, pages 42–54, New York, USA, 2003. ACM Press.
26. Annalisa Di Deo. *Modeling Spatial and Temporal Data in an Object-Oriented Constraint Database Framework*. PhD thesis, von der Fakultät IV-Elektrotechnik und Informatik der Technish Universität zur Erlangung des akademischem Grades, Berlín, 2002.
27. David Toman. SQL/TP: A temporal extension of SQL. In *Constraint Databases*, pages 391–399. Springer, 2000.
28. J. Warmer and A. Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley, 2 edition, 2003.
29. Andreas Petter, Alexander Behring, and Max Mühlhäuser. Solving constraints in model transformations. In *ICMT*, pages 132–147, 2009.
30. Ákos Horváth and Dániel Varró. Csp(m): Constraint satisfaction problem over models. In *MoDELS*, pages 107–121, 2009.
31. Michael Stonebraker, Dorothy Moore, and Paul Brown. *Object-Relational DBMSs: Tracking the Next Great Wave*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.
32. Juan Manuel Vara, Belén Vela, Verónica Andrea Bollati, and Esperanza Marcos. Supporting model—driven development of object—relational database schemas: A case study. In *Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations*, ICMT '09, pages 181–196. Springer-Verlag, 2009.
33. Paolo Atzeni, Paolo Cappellari, and Giorgio Gianforme. Midst: model independent schema and data translation. In *SIGMOD Conference*, pages 1134–1136, 2007.