Developing a labelled object-relational constraint database architecture for the projection operator

María Teresa Gómez-López, Rafael Ceballos, Rafael M. Gasca, Carmelo Del Valle

University of Seville, Language and Computer Systems Department, ETS Ingeniría Informática s/n, Avd de Reina Mercedes, 41012 Sevilla, Spain

A B S T R A C T

Current relational databases have been developed in order to improve the handling of stored data, however, there are some types of information that have to be analysed for which no suitable tools are available. These new types of data can be represented and treated as constraints, allowing a set of data to be represented through equations, inequations and Boolean combinations of both. To this end, constraint databases were defined and some prototypes were developed. Since there are aspects that can be improved, we propose a new architecture called *labelled object-relational constraint database* (LORCDB). This provides more expressiveness, since the database is adapted in order to support more types of data, instead of the data having to be adapted to the database. In this paper, the projection operator of SQL is extended so that it works with linear and polynomial constraints and variables of constraints. In order to optimize query evaluation efficiency, some strategies and algorithms have been used to obtain an efficient query plan.

Most work on constraint databases uses spatiotemporal data as case studies. However, this paper proposes model-based diagnosis since it is a highly potential research area, and model-based diagnosis permits more complicated queries than spatiotemporal examples. Our architecture permits the queries over constraints to be defined over different sets of variables by using symbolic substitution and elimination of variables.

Keywords:
Constraint database architecture
Model-based diagnosis
Knowledge-based design methodologies and tools
Query language
Query evaluation

1. Introduction

Relational databases have numerous utilities for classic data such as *Integer*, *Date* and *String*. However, there are types of data that cannot be handled nor stored in current relational databases in an easy way, thereby it is necessary to look for other alternatives. These complex types of data are those where the information is very large or even infinite, since it describes the relation between a group of variables. Fig. 1 shows two different examples where the values of variables x and y are infinite, hence it is necessary to define a way to store a representation of this data. Fig. 1a can be represented using the four corners of the rectangle, but the constraint of Fig. 1b cannot be stored in this way. If only some values of Fig. 1b are selected, the information stored in the database will be neither complete nor accurate. If the information is stored as constraints, all the values of x and y will be stored in an intensional way, since the representation and visualization of the information are more straightforward than if the extensional values are stored. These constraints make it easier and more precise to handle data that represent relations between variables which cannot be stored in a classic relational database.

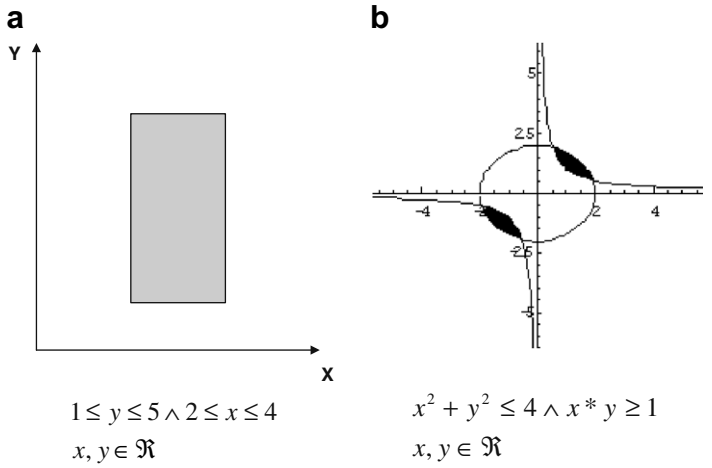

Fig. 1. Examples of constraints.

The applications in which the information can be represented by constraints are very common in engineering and industrial activities, for example in CAD/CAM (computer-aided design and manufacturing) [53], GIS (geographic information systems) [48], scientific data [24], and medical data [14], among others. In these cases, there is a great quantity of data that cannot be stored in an extensional way because infinite tuples of values would be produced. A method for handling the information of these applications has been approached by constraint databases (CDBs), since CDBs allow the storage of constraints in a database. The problem is that there are no solutions that allow constraint data to be handled with the same versatility as classic information. By using constraints to store the relation between variables it is possible to represent both intensional and extensional information. It entails storing constraints directly in a database and offering a way to answer queries about constraints whilst simultaneously permitting classic data to be accessed.

Although there are important and interesting proposals in the CDB field, there are aspects that can be improved in relation with the projection operator and the execution time for its evaluation. We have improved some ideas, such as that introduced by Veltri [55]. Veltri proposed storing the box consistency of the variables for each constraint to improve the computational time, but our proposal is more general since the solution of Veltri works only with linear constraints, and the box consistency is stored combined with the constraints. Our solution extends the use of polynomial constraints and proposes another way to store the box consistency in a classic field, as explained in the following sections. For this reason, an extension of SQL for the treatment of constraints as a new data type is presented together with the architecture which supports it. This paper presents a new architecture named labelled object-relational constraint database (LORCDB) which stores the constraints as objects in a relational database which are labelled depending on their types. These labels determine which technique is more appropriate in order to evaluate the queries. Moreover, different optimizations are presented in order to improve the computational complexity.

There are five primitive operations defined in relational algebra for classic information. These operations are: selection, projection, Cartesian product, union and difference. A selection over a relation obtains a horizontal subset of tuples where a condition is satisfiable, thereby if the relation is formed by constraints it is necessary to adapt the type of conditions over constraints and the method of evaluation. For union and difference operations two relations are combined vertically into a new relation. For both operators it is necessary to compare the different tuples to know if there are shared solutions and to adapt them for constraint information. Finally, the Cartesian product makes a concatenation between two relations, and its behaviour is equal to that in classic relational algebra.

In this paper, only the projection operator is redefined for constraint data, since it has specific and significant problems that have to be analysed. The rest of the operators will be studied in future work. The projection operator obtains a vertical subset of attributes of a relation. When constraint information is stored in a relation, it means that a set of the variables that form the constraints can be obtained. These variables can be represented in a symbolic way as a new relation or as the possible values that these variables represent in a extensional way.

In order to evaluate the different queries about constraint information related to the projection, sometimes it is necessary to create constraint problems as constraint satisfaction problems, constraint optimization problems and symbolic variable-elimination problems. These problems are created dynamically according to the query and they are solved using different techniques: Gröbner Bases algorithm; cylindrical algebraic decomposition algorithm, constraint satisfaction problem solver or constraint optimization problem solver. In the proposed architecture all these techniques are combined in order to redefine the projection operator for CDBs. In order to develop this architecture several techniques that allow the evaluation of the queries have been used, and since these techniques can be computationally difficult, some improvements are presented such as the indexation between variables and constraints; labelling the constraints according to their types, in order to combine

them in an efficient way; looking for the constraints related to the query; using heuristics to solve constraint satisfaction problems; and using relational and object-oriented paradigms.

Although our architecture can be used for any type of problem which involves linear or polynomial constraints, in this work model-based diagnosis is employed since it has many industrial utilities, such as with hardware and software components, industrial plants and vehicles. Another reason for using model-based diagnosis is that very complex problems can be defined, where many variables can participate. Hence, it implies that very different types of queries can be defined which require different procedures for their evaluation, while spatiotemporal problems are defined over only four dimensions (height, width, length and time). This work is organised as follows: Section 2 presents some definitions about constraint databases and analyses previous work on CDBs. Section 3 gives some definitions necessary for the explanation of model-based diagnosis, which is the case study used in this paper. Section 4 shows a new definition of CDB, the query language to create and query a LORCDB and the proposed architecture. Section 5 explains how a query is evaluated. Section 6 presents some examples to show how the models are created in order to evaluate the queries by using the different techniques for the different types of projection. Section 7 presents some examples of queries and their average evaluation times, and shows how the use of our architecture improves the computation time for the different projections over constraints. Finally, conclusions and future work are presented.

2. Background

Constraint databases were initially developed in 1990 with a paper by Kanellakis et al. [37], and were extended through research on a query language which is a subset of Prolog from the syntactical point of view (Datalog [9]) and through constraint logic programming (CLP) [35,36]. However, Datalog and Prolog differ in their semantics, whilst Prolog uses a depth-first search method, Datalog uses breadth-first search strategy which returns answer sets. These paradigms were used to define the first constraint databases, and created a new research area [46,47]. The basic idea behind the CDB model 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.

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

2.1. Constraint databases definitions

Before starting the explanation of some of the most important prototypes developed in the constraint databases, some modified definitions from [20] are necessary.

Definition 2.1 (*Vocabulary*). A vocabulary Ω consists of three sets: a set $\mathscr{F}$ of function symbols, a domain of values $\mathscr{D}$ and a set $\mathscr{P}$ of predicate symbols. The sets of functions and predicates have an associated arity. For example the vocabulary $\Omega = (\{+,\cdot\}, \{0,1\}, \{<\})$ has two function symbols of arity two $(+,\cdot)$, one predicate symbol with arity two $(<)$, and two constant symbols 0 and 1.

Definition 2.2 (*First-order logic over Ω).* Terms are defined inductively as a variable, or a value $v \in \mathscr{D}$, or $f(t_1,\ldots,t_n)$ where $f \in \mathscr{F}$ is of arity n, and $t_1,\ldots,t_n$ are terms.

Definition 2.3 (*Atomic formula*). This has the form $t \equiv t'$ or $p(t_1,\ldots t_n)$, where t, t', $t_1,\ldots,t_n$ are terms, and $p \in \mathscr{P}$ is an n-ary predicate.

A *formula* can be an *atomic formula*, or a Boolean combination of atomic formulas. In our case, if φ and ψ are formulas, then $(\varphi \wedge \psi)$ and $(\varphi \vee \psi)$ are also formulas.

Definition 2.4 (*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 definitions can be expressed with the following grammar:

```
Constraint:= Atomic_Constraint BOOL_OP Constraint
   | Atomic_Constraint
   ;
BOOL_OP:='∨'| '∧'
   ;
Atomic_Constraint:= function PREDICATE function
   ;
function:= Var FUNCTION_SYMBOL function
   | Var
```

```
   | Value
   ;
PREDICATE:=' ='| '<'|
   ;
FUNCTION_SYMBOL:='+'| '-'| '*'
   ;
```

The constraint types permitted in our architecture are based on [36,11], and are:

- *Linear equality constraints:* These are constraints over the vocabulary $\Omega = (\{+\}, \{\mathscr{D}\}, \{=\})$, where $\mathscr{D}$ can be $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{R}$. Such constraints are of the form $p = 0$, where p is a linear function of the form $a_1x_1 + \cdots + a_jx_j + a_0$, where a_i is a coefficient belonging to the domain $\mathscr{D}$, and x_i is a variable.
- *Polynomial equality constraints:* These are constraints over the vocabulary $\Omega = (\{+, \cdot\}, \{\mathscr{D}\}, \{=\})$, where $\mathscr{D}$ can be $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{R}$. Such constraints are of the form $p = 0$, where p is a polynomial with coefficients belonging to the domain $\mathscr{D}$.
- *Linear inequality constraints:* These are constraints over the vocabulary $\Omega = (\{+\}, \{\mathscr{D}\}, \{<\})$, where $\mathscr{D}$ can be $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{R}$. Such constraints are of the form $p > 0$ or $p \leqslant 0$, where p is a linear function of the form $a_1x_1 + \cdots + a_jx_j + a_0$ with coefficients belonging to the domain $\mathscr{D}$.
- *Polynomial inequality constraints:* These are constraints over the vocabulary $\Omega = (\{+, \cdot\}, \{\mathscr{D}\}, \{<\})$, where $\mathscr{D}$ can be $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{R}$. Such constraints are of the form $p > 0$ or $p \leqslant 0$, where p is a polynomial with coefficients belonging to the domain $\mathscr{D}$.

Definition 2.5. Constraint database modelGiven a vocabulary Ω:

- *A constraint k-tuple*, with the variables $x_1, \ldots, x_k$, over Ω, is a finite conjunction $\varphi_1 \wedge \cdots \wedge \varphi_N$, where each φ_i $(1 \leqslant i \leqslant N)$ is an Ω-constraint. The variables of each φ_i are all contained in $x_1, \ldots, x_k$.
- *A constraint relation of arity k, over* Ω, is a finite set $r = \{\psi_1, \ldots, \psi_M\}$, where each ψ_i $(1 \leqslant i \leqslant M)$ is a constraint k-tuple over the same variables $x_1, \ldots, x_k$. The corresponding formula is the disjunction $\psi_1 \vee \cdots \vee \psi_k$. This formula is represented in a relation as different tuples, where each ψ_i forms a *constraint k-tuple*.
- A constraint database is a finite collection of constraint relations. An example of a constraint relation is presented in Fig. 2.

2.2. Related work

There are several important proposals for implementing and building prototypes for CDBs. The most important proposals are analysed below:

- *DISCO* [8] (DATALOG with Integer Set COnstraints) is a CDB system which implements Datalog with Boolean constraints on variables that range over finite and infinite sets of integers. The constraints in DISCO are stored in a file using facts and rules, thereby the advantages of relational databases are not used, despite the fact that a lot of research about relational databases have been published over the years, such as in the design of large-scale databases and concurrent access. The syntax of DISCO is very similar to that of Datalog, and very different from the standard SQL. DISCO is presented as an extensible system where the Boolean algebra can be modified by replacing some operators in a non-complicated way.
- *MLPQ/PReSTO* [49,48] presents a combination of MLPQ (Management of Linear Programming Queries) and PReSTO (Parametric Rectangle SpatioTemporal Object). MLPQ is a system to manage and query linear constraints in a CDB. It allows Datalog queries and the addition of operators over linear functions. PReSTO enables query systems which change over time, with similar semantics to that of classic relational algebra. Although both present a similar SQL syntax, they actually use a plain text to store the information and a query transformation process into Datalog.

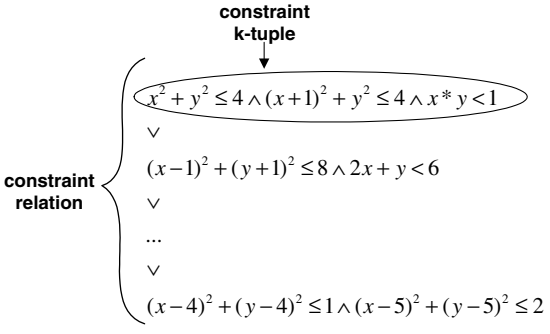

Fig. 2. An example of a constraint relation.

- *DEDALE* [31] is an implementation of CDBs based on linear constraint models. DEDALE proposes a language to query CDBs, which permits information to be obtained by using a graphical interface to show the results. In order to represent the constraints, DEDALE uses the object-oriented paradigm, which is an appropriate way to represent complex data. The disadvantage in this proposal is that all the information is stored as objects, therefore, does not take advantage of relational algebra.The syntax of DEDALE is very different from standard SQL, and therefore, any user of the system must learn another language, which is a great disadvantage.
- *CCUBE* [5] (constraint comprehension calculus) is the first constraint object-oriented database system. The CCUBE system was designed to be used for the implementation and optimization of high-level constraint object-oriented query languages. The CCUBE data manipulation language, constraint comprehension calculus, is an integration of constraint calculus for extensible constraint domains within monoid comprehension. CCUBE serves as an optimization-level language for object-oriented queries. The data model for the constraint calculus is based on Constraint SpatioTemporal (CST) objects. CCUBE guarantees polynomial time data complexity whose implementation uses the linear programming package CPLEX developed by Bixby et al. [4]. CPLEX allows both integer programming and very large linear programming problems to be solved.
- *CQA/CDB* is a CDB prototype whose constraints are linear over rational coefficients [27,28]. This solution uses only linear constraints in order to represent and process the information, since it studies the spatiotemporal area, whose data can be approximated with linear constraints. The constraint tuples are represented by a matrix of coefficients. In this prototype, queries are based on relational operators, and look similar to SQL; however, there are numerous syntactic and semantic differences to the standard. CQA/CDB does not allow new constraints to be inferred by using the constraints stored in the CDB.

The different characteristics are shown in Table 1. The main disadvantages of the proposals mentioned are:

- None of these prototypes offers a versatile solution for any type of application, neither do they offer a general solution independent of the nature of the problem domain. Most of these prototypes are developed to work with spatiotemporal data such as [21,54], since the domain is specific. In this paper, model-based diagnosis is used since it has some characteristics which are different from the spatiotemporal information.
- There are many query languages whose syntax are very different to SQL, whereby the user needs to learn a new language. Sometimes this syntax is focused on the application of CDBs, frequently in the spatiotemporal domain.
- Proposals, such as MLPQ, do not use relational databases, therefore, they have worse execution times than solutions based on relational databases. Some of these proposals are based on deductive databases whose disadvantages of consistency were analysed in [42]. A lot of experiences and theories in relational databases have improved the retrieval of data using indexation, clustering and hashing. All of them have been included in commercial relational databases managements, but not when deductive databases are used.
- These proposals only handle linear constraints, since, for spatiotemporal problems, this type of constraint is good enough to approximate the problem, and solving linear constraint problems is more efficient than for polynomial constraints. This approximation is inappropriate for other applications where more precision is necessary.
- None of these prototypes defines *constraints* as a new type of data, and although some of them use the object-oriented paradigm, they do not distinguish between discrete and continuous information, and hence are unable to exploit the relational algebra in any of these aspects.
- These proposals are not focused on handling several variables simultaneously. Although work such as [20,39,10] analyses the projection operator over the variables of a constraint by using quantifier elimination, there are no proposals that define the projection over a set of constraints with different variables which generate new constraints by using variable substitution techniques. This type of projection is necessary in applications such as model-based diagnosis, whose details are explained in the next section.

3. Model-based diagnosis: a new application domain for CDB

Most work on constraint databases uses spatiotemporal data as case studies, however, this paper proposes model-based diagnosis since it is a highly potential research area. The use of model-based diagnosis permits the enrichment of both the

Table 1
Characteristics of the CDB prototypes

	Logic model	Logic Progr.	SQL syntax	Constraint type	Any application	Multiple variables
MLPQ/PReSTO	File	Datalog	Yes	Linear	No	No
DISCO	File	Datalog	No	Linear	No	No
DEDALE	BDOO	No	Yes	Linear	No	No
CCUBE	BDOO	No	Yes	Linear	No	No
CQA/CDB	?	No	Yes	Linear	No	No
LORCDB	BDOR	No	Yes	Linear and polynomial	Yes	Yes

type of constraints stored and queries about them, by defining problems that are more complicated than spatiotemporal examples.

Fault detection and identification of faulty components are very important from the strategic point of view of companies due to the economic demands and environment conservation required to remain in competitive markets. This paper proposes storing and handling all the information related to the model-based diagnosis process in a CDB, and also helps in fault detection. In engineering applications, such as model-based diagnosis, the storage of these data and the query processing are often overlooked. Although spatiotemporal problems are the normal field of study in CDBs, model-based diagnosis is a very good example since it is possible to use a great number of variables while spatiotemporal problems are defined over only four dimensions (height, width, length and time). Furthermore, the use of model-based diagnosis also proves that CDBs can be used in other types of applications.

Model-based diagnosis allows the identification of failures in a system, and by using CDB technology it is possible to make information of the models readily available. Model-based diagnosis began in the 1980s with the DX proposal in [16,43,38]. These papers were developed in order to discover the discrepancies between the observed and correct behaviour of a system.

Constraint satisfaction problems have sometimes been used in discrete model-based diagnosis [52,50]. The relation between the components is equal to the network of related constraints defined in [40], although in our proposal the components have been modelled as constraints and the links between the components are their shared variables.

In order to clarify the necessity of using the projection operator in some scenarios, the following definitions on model-based diagnosis are required. These definitions can be explained by means of a simple example shown in Fig. 3. The example presents a set of components, multipliers (*Mi*), and adders (*Ai*), which work together and where the sensors are located in the variables *a*, *b*, *c*, *e*, *d*, *f* and *g*.

Definition 3.1. (*System polynomial model (SPM)*). This can be defined as a finite set P of linear or polynomial equality constraints which determines the system behaviour. This set is created by using the relation between the non-observable system variables (V_{nob}) and the observable variables (V_{ob}) which are directly obtained from sensors which are assumed to work correctly. An SPM is formed by the tuple $\{P, V_{\text{ob}}, V_{\text{nob}}\}$.

Definition 3.2. (*Context set (CS)*). Any subset of components which composes the system. There are $2^{nComps} - 1$ possible context sets, where *nComps* is the number of components of the system.

Definition 3.3. (*Context analytical redundancy constraint (CARC)*). A constraint derived from the system in such a way that only the observable variables are related. By using CARCs, the diagnosis process is made easier, since a new system with only observable variables is created. In the example of Fig. 3 the CARCs are:

- CARC_1 $\{f = a * c + b * d\}$, generated with the constraints of the components $\{M_1, M_2, A_1\}$.
- CARC_2 $\{g = b * d + c * e\}$, generated with the constraints of the components $\{M_2, M_3, A_2\}$.
- CARC_3 $\{f - g = a * c - c * e\}$, generated with the constraints of the components $\{M_1, M_3, A_1, A_2\}$.

Definition 3.4. (*Observational model (OM)*). A set of tuples of values for the observable variables of the system. The CARCs are formed of only variables with sensors, meaning that the values of an OM can be substituted in a CARC. If a CARC is satisfiable for an OM and there is no fault compensation, this means that the components associated with the constraint work correctly. Otherwise, if the CARC is unsatisfiable, then it is known that at least one component of the CS is not working correctly.

For example, for the OM $\{a = 1, b = 2, c = 2, d = 3, e = 2, f = 9, g = 11\}$, CARC_1 and CARC_2 are unsatisfiable. It means that at least one component of $\{M_1, M_2, A_1\}$ and another of $\{M_2, M_3, A_2\}$ are failing. A solution of the diagnosis in this case is that the component M_2 is failing.

A system composed of components that can be handled by the projection operator of the relational algebra adapted to the constraints, has the following characteristics:

- Obtaining some values of a correct behaviour of the system is similar to performing a projection where the values of some of the variables of the constraints are obtained.
- To obtain the CARCs of a system means obtaining the same constraints represented only by a set of variables. This idea is similar to that of the projection operator, where only a set of attributes are obtained. The main difference is that in CDB the attributes can be variables.

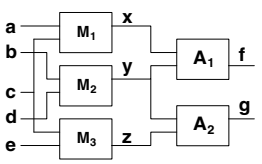

Fig. 3. Fault diagnosis toy example.

- Deducing whether a constraint is satisfiable is similar to obtaining a constraint using a projection where the variables are instantiated.

In the following sections, the methodology for adapting the projection operator for constraints and their variables is explained. Using our SQL extension over constraints makes it possible to obtain several constraint models depending on the query, which can be compared to real values in order to perform the diagnosis of the system. The location of sensors defines which variables are observable, while the remaining variables are non-observable.

4. A new definition of CDB

In order to differentiate between the intensional or extensional information in a relation of a CDB, the definition of CDB is changed in this paper as follows.

Definition 4.1 (*Revision of the CDB model).*

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

In Fig. 4, the relationship between a constraint relation and a constraint k-tuple is presented. A relation has classic attributes if and only if:

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

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

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 classic attribute, since this variable follows the relational algebra defined by Codd.

An example is presented in Fig. 5, where the constraint relation is composed of a constraint attribute and two classic attributes. In the example, there are two variables (x and y) that appear in all the tuples which have an equality relation with a constant, and do not appear in the rest of the attributes.

- Therefore, a *constraint database* is a finite collection of constraint relations composed of *classic* and constraint attributes.

Derived from the previous definitions, it is necessary to enlarge the types of attributes for CDBs. We define three types of attributes:

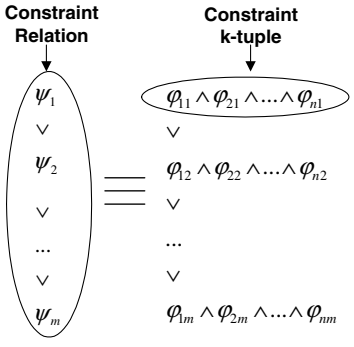

Fig. 4. Representation of constraint k-tuple and constraint relation.

- *Classic attribute* (at_i)*:* at_i belongs to the n-tuples of a relation $(1 \leqslant i \leqslant n)$, where at_i is of a type permitted by standard SQL, for example *Integer*, *String*, *Date*, etc. This type of attribute can be represented as a constraint such as $\{at_i = \text{constant}\}$, since relational databases only permit attributes with atomic values.
- *Constraint attribute* (at_i^c)*:* at_i^c belongs to the n-tuples of a relation where at_i^c is of *constraint* type defined as a *constraint k-tuple* with the variables $v_1 \cdots v_k$, over a vocabulary Ω. A constraint attribute represents a relation between the set of variables $v_1 \cdots v_k$. For our proposal a constraint relation is formed by classic and constraint attributes.
- *Constraint-variable attribute* $(at_i^c \cdot v_j)$: v_j is a variable which belongs to a constraint attribute at_i^c. This attribute is represented by:

  ⟨Constraint_column_name⟩·⟨Variable_name⟩.

  where ⟨Variable_name⟩ is not an attribute, but is a variable of a constraint stored as a value inside the column *Name* of the table *Variables*. A column of the database (constraint attribute) can have several constraints, and these constraints can be defined over different variables.

In order to give support to the definitions of constraint k-tuple, constraint relation and CDB, and to enable the use of the projector operator over the new attributes, we propose a new language and an architecture called *labelled object-relational constraint database* (LORCDB).

4.1. CORQL: constraint object-relational query language

This section shows the syntax of the language to create, query and fill a LORCDB, whereas the following sections present the details about the architecture. Some implementation decisions have been proposed for the improvement of average evaluation query time of the queries, for the accessibility to information and for the flexibility in the storage when dealing with different types of data.

In this work, the projection operator is revised since it is a vertical subset of a relation. For CDBs the projection can obtain a subset of constraints or a subset of variables of a set of constraints, which makes it necessary to define this type of query a new way. The semantics of SQL has been extended whilst changing the syntax of the queries as little as possible. In order to facilitate the handling of constraints by inexpert users, we propose a new language CORQL (constraint object-relational query language), developed to facilitate the relation between the user and the system for the projection operator. Since one of the most important aims of our proposal is to make the utilisation of constraints transparent to the user, therefore, the language CORQL maintains very similar syntax to SQL. CORQL is a superset of SQL, where new syntactical constructors are added in order to handle constraint data. This expansion is based on the addition and querying of constraint information.

4.1.1. Creating a LORCDB

In order to create a LORCDB the following sentence is used:

```
CREATE LORCDB⟨database_name⟩
```

The constraints used in the architecture are numerical, and are stored as objects indexed by the variables contained within, hence, when a LORCDB is created, three auxiliary tables are also automatically created (*Constraints*, *Variables* and *Constraints/Variables*) which relate each constraint with its variables. These tables, shown in Fig. 6, improve the computation time of obtaining the constraints related to the variables of a projection. There are studies [30] that improve the compilation time for the query which handles the constraints, whereas our proposal focuses on the improvement in running time. These tables make it unnecessary to study all the constraints for a query. The *Constraints/Variables* table can also store the minimum and maximum value of each variable for every constraint, which is the box consistency for each variable [29] obtained by creating and solving a COP combined to interval arithmetic [3]. These tables are not directly accessible to the users, however, these tables implicitly change according to the constraints added to or removed from the LORCDB. The table *Constraints* stores the *idConstraint*, which is the object identification (OID) generated by the system, and the *Label* according to the type of constraint, in order to decide which technique will be used to handle the constraint. The table *Variables* stores the names of the

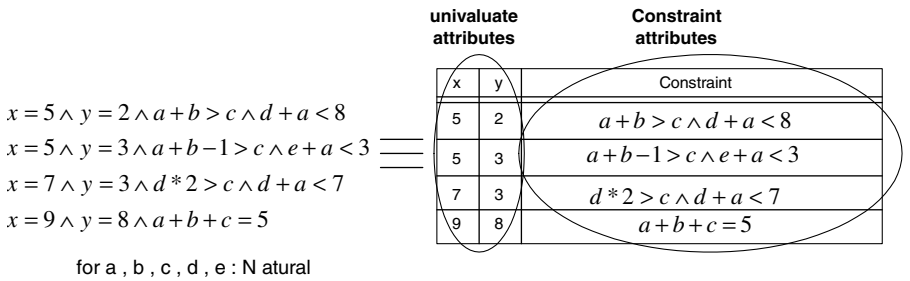

Fig. 5. Example of constraint relation with constraint k-tuples.

variables, their identification and their type (Integer, Natural or Float). The users cannot see nor modify the indexes in order to prevent the database from changing information without control. These internal tables are just used to speed up the evaluation of the queries in order to obtain the constraints related to each query. These indexes are updated when constraints are added, and they are created and stored in an implicit way in order to locate constraints more quickly and efficiently.

4.1.2. Creating and inserting information into a table

As our proposal tries to modify SQL syntax as little as possible, the syntax is not modified at all for the creation of a table. The only change is that it is now possible to create constraint fields as *constraint* type. It is also possible to create a table with several columns of *constraint* type.

For the example shown in Fig. 3, the sentence is:

```
CREATE TABLE Component(IdComponent Integer,
   Name String, Behaviour Constraint)
```

If a field of a table has been created as *constraint* type, it is possible to add constraint information to this table. Therefore, if the user tries to add information in a field of an incorrect type, an error will be produced as in a common relational database. The checking of the type is performed as for usual types.

The *constraint* objects are stored as syntactic trees, where the internal nodes can be comparator operators (=, <, $\leqslant$, > or $\geqslant$), Boolean operator ($\wedge$), or arithmetic operators (+, − or $*$). The leaves of the trees are the variables and constants of the constraint. This means that any logical combination of equations and inequations of constraints, as defined in the grammar of Section 2, can be represented as a syntactic tree. When a constraint is added it is also possible to define the type and range of its variables. If the range of a variable is not specified, by default it is defined as {Integer.MIN_VALUE..Integer.MAX _VALUE} for Integer variables, {0..Integer.MAX_VALUE} for Natural variables, and {Float.MIN_VALUE..Float.MAX_VALUE} for Float variables, values provided by the language *Java*™ which is used to implement CORQL.

The syntax to insert a constraint (according to Definition 2.4) is:

```
{⟨Constraint⟩,
   (⟨type⟩ [⟨Min_Value⟩..⟨Max_Value⟩] ⟨variable⟩, ...)}
```

In order to store the example shown in Fig. 3 in a LORCDB, a sentence can be:

```
INSERT INTO Component(IdComponent, Name, Behaviour) VALUES
   (101,'A1', { x + y = f, (Float 0..100 x, Float 3..50 y, Float f)})
```

There are four types of constraint labels: linear equality (LinEQ), linear inequality (LinIEQ), polynomial equality (PolEQ) and polynomial inequality (PolIEQ). The labels help to decide which tool is more appropriate to solve the constraint problem that will be created depending on the query.

4.1.3. Querying a labelled object-relational CDB

An attribute of a relation can be a constraint, and a constraint is a relation of variables. Thereby it is necessary to analyse the new queries permitted by our architecture, and hence a new type of data and a new way to query this data are proposed here. According to the new attributes, we propose an extension of the functionality of the projection operator when the information involved in the query is a constraint. This operator is analysed below, and we study how it is affected when working with constraint information.

The projection operator has the signature $R_1 = \pi_{(a_1,\ldots,a_n)}(R_2)$, where R_1 and R_2 are relations and $(a_1, \ldots, a_n)$ are attributes. Relation R_1 is a vertical subset of the relation R_2, and is composed of only the specified attributes and without duplicated tuples. Some examples for model-based diagnosis can be the following queries, that use the projection operator for the different types of attributes:

(1) $\pi_{(at_1 \cdots at_n)}(R)$. What is the identifier of component *A*1? This type of query is the one efficiently analysed by classic relational algebra.
(2) $\pi_{(at_1^c \cdots at_n^c)}(R)$. Obtain the attribute *Behaviour* of relation *Component*, where *Behaviour* is of *constraint* type. As defined in Definition 4.1, a *constraint k-tuple* is formed by a conjunction of Ω-constraints, thereby it can only be formed by $\wedge$ as a Boolean operator. In order to represent the $\vee$ operator, the constraints will be stored in different tuples that represent the *OR* operator.
(3) $\pi_{(at_1^c \cdot v_1 \cdots at_1^c \cdot v_k \cdots at_n^c \cdot v_1 \cdots at_n^c \cdot v_k)}(R)$. Obtain 20 values of variables *a*, *c*, *d*, *e* and *f* for the attribute *Behaviour* of relation *Component*.

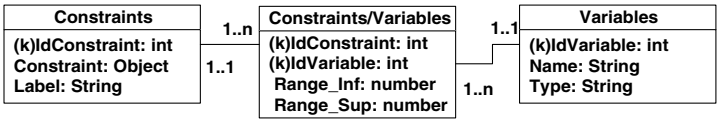

Fig. 6. Database tables to index constraints and variables.

It is only necessary to redefine the syntax of the projection over constraint-variable attributes ($at_i^c \cdot v_j$), and there are two different ways to obtain the evaluation of this type attribute, either in an extensional or in an intensional form. It will be possible to obtain tuples of satisfiable values of the variables, or to obtain the symbolic relation of the variables as new constraints inferred from the original constraints with only those variables that appear in the projection.

In order to carry on the symbolic projection over variables of constraints, a new type of query is defined. When a user of the LORCDB wants to obtain a set of variables in a symbolic way, the following syntax is used:

$$\texttt{SELECT CONSTRAINTS}(at_1^c \cdot v_1, \ldots, at_1^c \cdot v_k, \ldots, at_n^c \cdot v_1, \ldots, at_n^c \cdot v_k)\ \texttt{FROM}\ \langle \texttt{Table} \rangle$$

If it is possible, a new set of constraints formed by only the variables of the projection is inferred, using symbolic-elimination techniques based on existential quantifier elimination. This quantifier has been analysed in [45], where it is defined that a conjunction of constraints over the variables $x_1, \ldots, x_i, \ldots, x_n$ can be transformed into another constraint by eliminating x_i if and only if there is an S_* such that $S^* = \exists x_i S$.

In order to carry on the numeric projection over variables of constraints, two different queries are defined. When a user of the LORCDB wants to obtain a group of values of variables, the following syntax is used:

$$\texttt{SELECT VALUES[n]}(at_1^c \cdot v_1, \ldots, at_1^c \cdot v_k, \ldots, at_n^c \cdot v_1 \cdots at_n^c \cdot v_k)\ \texttt{FROM}\ \langle \texttt{Table} \rangle$$

where n is the number of tuples of values that the user wants to obtain for the variables of the projection, since it is possible that the number of solutions will be infinite.

Some aggregate functions are also revised for constraints in our proposal. An aggregate function takes a set of tuples (a relation) as an argument and produces a single value (usually a number) as a result. The main functions are *COUNT*, *SUM*, *AVG*, the *MIN* and *MAX*. For our language CORQL, only the *MIN* and *MAX* functions are extended in order to obtain the maximum or minimum value of a variable, whereas the rest of the aggregate functions will be studied in future work. The syntax of the *MIN* and *MAX* functions to obtain the minimum or maximum value of the variable v_j in the constraint attribute at_i^c is:

$$\texttt{SELECT [MAX|MIN] VALUE}(at_i^c \cdot v_j)\ \texttt{FROM}\ \langle \texttt{Table} \rangle$$

4.2. Computational techniques

In order to evaluate and obtain the outputs of the types of projection operator presented above, different techniques are used. These techniques are divided into two different families: symbolic techniques (Gröbner Bases and cylindrical algebraic decomposition), used in order to obtain new symbolic constraints; and numeric techniques (constraint satisfaction and constraint optimization solvers), used in order to obtain values of variables. Gröbner Bases are used for equality constraints, while cylindrical algebraic decomposition is used for quantifier elimination when inequality constraints are involved. Some of their characteristics are explained in the following subsections.

4.2.1. Gröbner Bases

One of the most promising schemes to solve systems of polynomial equality equations is Gröbner Bases. Gröbner Bases theory is the origin of many symbolic algorithms used to manipulate multiple-variable polynomials. This scheme is a generalization of Gauss's elimination of multivariable linear equations and of Euclides's algorithm for one-variable polynomial equations.

Using Gröbner Bases makes it possible to transform a set of polynomial constraints into a standard form. Let the set of polynomial equality constraints be in the form $P = 0$, hence Gröbner Bases theory allows an equivalent system $G = 0$ to be obtained, which has the same solutions as the original but is easier to solve. Gröbner Bases have better computational properties than the original systems.

An easy example of Gröbner Bases is: By using the equations $\{a + b = c, d + e = f, c * f = g\}$, obtain a new constraint where only the variables $\{a, b, d, e, g\}$ are presented. Here the solution is $\{g - (a + b) * (d + e) = 0\}$.

The Buchberger algorithm [7] was the first algorithm for computing such Gröbner Bases. The first improvement was related with strategies during Gröbner computation (F4 [23]). Another open issue was to remove useless computations, which was precisely the goal of Faugère [22] in order to give a theoretical and practical answer.

In general, the complexity of the Gröbner Bases algorithm is high, exponential according to the number of variables in the worst case, however, it is the best known method for symbolic substitution. For this reason, in this paper, we propose an algorithm which uses Gröbner Bases the fewest number of times possible.

4.2.2. Cylindrical algebraic decomposition

The quantifier elimination for inequality constraints is based on cylindrical algebraic decomposition. It is a technique used in the study of topological properties of semi-algebraic sets, which decomposes the space into a number of regions,

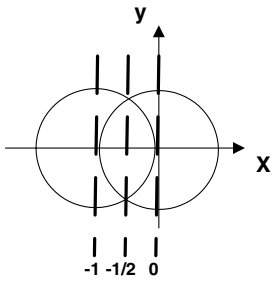

Fig. 7. Cylindrical algebraic decomposition example.

in each of which the equations and inequalities have the same sign (positive or negative). In this work, cylindrical decomposition is used to eliminate variables of a group of polynomial inequality constraints. More details can be found in [1,6,2].

The cylindrical algebraic decomposition returns inequality constraints whose borders involve algebraic functions. In order to explain these ideas, the example shown in Fig. 7 is used.

In the example of Fig. 7, the following constraints are represented:

$$(x^2 + y^2 < 1) \text{ and } ((x+1)^2 + y^2 < 1).$$

The first constraint is represented by a circumference centred at (0,0) of radius 1, and the second constraint by another circumference, also of radius 1, but centred at $(-1,0)$.

If cylindrical algebraic decomposition is performed over variable x, then the cells shown in the figure as discontinuous lines are created. These divisions are formed depending on the value of x where both constraints have the same y value, which leads us to perform an analysis of these points. In this case the following new constraints are obtained:

$$-1 < x < -1/2 \wedge -\sqrt{1-x^2} < y < \sqrt{1-x^2}$$
$$\vee$$
$$-1/2 < x < 0 \wedge -\sqrt{-2-x^2} < y < \sqrt{-2x-x^2}.$$

The quantifier elimination over variable y can separate the part of the constraint where only x is involved, and yields:

$$-1 < x < 0.$$

These constraints define the intersection of the original constraints. The major problem with cylindrical algebraic decomposition is the complexity, since the complexity of the algorithm is double exponential according to the number of cells. This problem has been analysed in order to improve the computational time [12,6,13,51]. In the same way as when Gröbner Bases are used, an algorithm is developed in order to avoid using cylindrical algebraic decomposition when it is not necessary. Another problem has to be considered when the quantifier elimination is used, since it is not always possible to obtain a solution over integer or natural domains [33].

4.2.3. Constraint satisfaction problems and constraint optimization problems

Constraint satisfaction problems (CSPs) represent a reasoning framework consisting of variables, domains and constraints. Formally, it is defined as a triple $\langle X,D,C\rangle$ where $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables, $D = \{d(x_1), d(x_2), \ldots, d(x_n)\}$ is a set of domains of the values of the variables, and $C = \{C_1, C_2, \ldots, C_m\}$ is a set of constraints. Each constraint C_i is defined as a relation R on a subset of variables $V = \{x_i, x_j, \ldots, x_k\}$, called the *constraint scope*. The relation R may be represented as a subset of the Cartesian product $d(x_i) \times d(x_j) \times \cdots \times d(x_k)$. A constraint $C_i = (V_i, R_i)$ specifies the possible values of the variables in V simultaneously in order to satisfy R. Let $V_k = \{x_{k_1}, x_{k_2}, \ldots, x_{k_l}\}$ be a subset of X, and 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 the variables in V_k. An instantiation is a solution only if it satisfies the constraints C.

Usually, to solve a CSP, a combination of search and consistency techniques is used [17]. The consistency techniques remove inconsistent values from the domains of the variables during or before the search. Several local consistency and optimization techniques have been proposed as ways of improving the efficiency of search algorithms.

A constraint optimization problem (COP) is a constraint satisfaction problem defined as a triple $\langle X,D,C\langle$ and an objective function f that has to be optimized (maximised or minimised):

$$f : \{d(x_1) \times d(x_2) \times \cdots \times d(x_n)\} \rightarrow \mathscr{S}_t \quad \text{where } (\mathscr{S}_t, \leqslant) \text{ defines the total order.}$$

For example, in a COP where an objective is maximised, $\mathscr{A}$ is a solution for the COP if and only if $\mathscr{A}$ is a solution of the CSP and no solution of $\mathscr{A}'$ exists where $f(\mathscr{A}') > f(\mathscr{A})$. Therefore, the best solution of the CSP for a goal is considered a solution of the COP [15,34].

4.3. Labelled object-relational CDB architecture

Here, the term architecture is used to refer to a set of software components which work with constraints. The computational techniques presented above contribute towards the definition of the new architecture (labelled object-relational constraint database) which permits constraint information to be processed by querying classic attributes, constraint attributes and constraint-variable attributes in order to create and solve constraint problems dynamically. This architecture is designed to improve the computational complexity in the query evaluation task, by combining the advantages of relational databases, object-oriented paradigms, indexation of variables and box consistency for each variable and constraint, thereby giving a holistic solution as a proposal.

The architecture follows a pattern formed by different layers, as shown in Fig. 8. It has four main subsystems: interface, query-driven model, computational processing and data layer:

- The *interface* layer reads the sentences from the user and presents the result of the query. The submodule *Lexical/Syntactic Analysis Filter* checks if the query is correct, as explained in Section 4.1. This module also selects those queries related to constraints by leading the query to *query-driven model*, and filters out the classic queries, in which case the database can offer the solution and no other module is necessary. This means that in a LORCDB, not only is it possible to store the same information as in a relational database, but it is also possible to store constraints as objects.
- The *query-driven model* layer obtains the constraints related to the query, and builds a model that will be solved by the appropriate tool. The submodule *search engine for the related constraints* obtains the constraints of the database that are needed to evaluate the projection depending on the type of query, as explained in Section 5. This submodule is crucial since it allows the creation of constraint problems with only those constraints necessary, thereby improving the computational evaluation time. Once the related constraints are known, the submodule *Build Models depending on the type of Query* works as an adapter by creating different types of models as explained in Section 6. The types of models are divided into two families, depending on the technique necessary for their solution: *numeric resolution* or *symbolic resolution*. The submodule *numeric resolution* deals with queries whose solutions are numeric values (when the projection uses the word `VALUES`), while the submodule *symbolic resolution* works with symbolic techniques for queries whose answers are new constraints inferred from the stored constraints (when the projection uses the word `CONSTRAINTS`). At the same time, the module *numeric resolution* has two submodules in order to build *constraint satisfaction problems* (if a set of values has to be obtained) or *constraint optimization problems* (if only the minimum or maximum value of a variable has to be known). The module *symbolic resolution* has two submodules that are used depending on the type of constraints related to the query. These submodules are: *Gröbner Bases symbolic technique* for equality polynomial constraints, and *cylindrical algebraic decomposition* for inequality polynomial constraints.

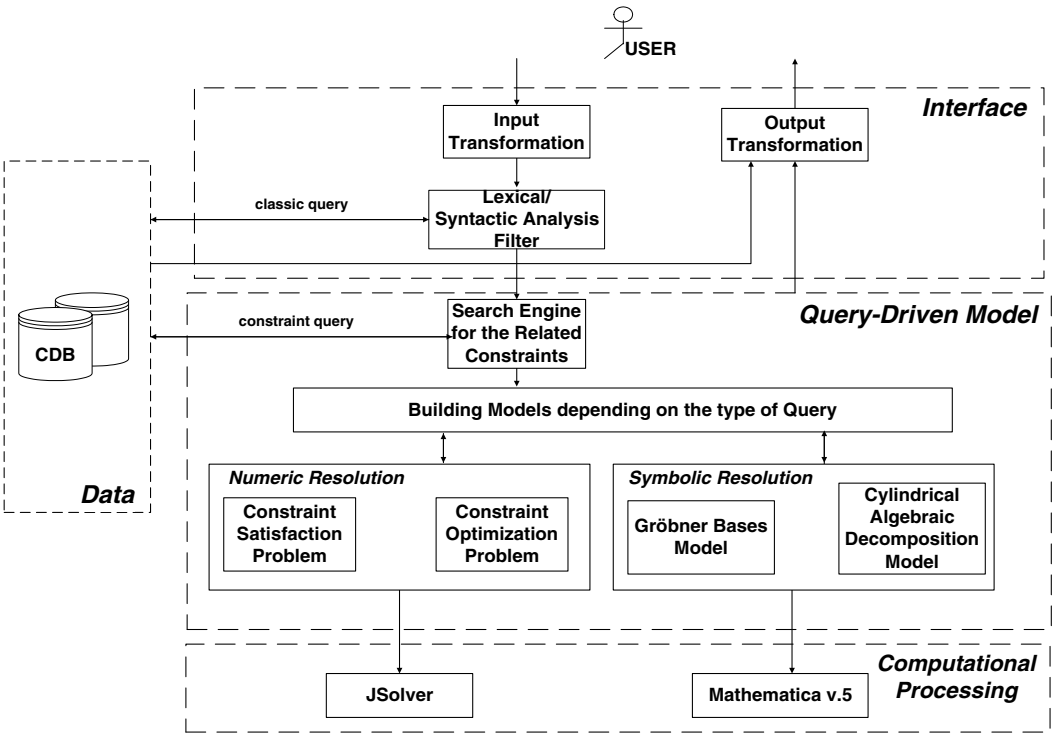

Fig. 8. Labelled object-relational CDB architecture for projection.

- The *Computational Processing* layer solves the different types of problems created in the *query-driven model* by using different tools. In our case *ILOG™) JSolver* [41] and *Mathematica™)v.5* [44] are used as examples, although it is possible to include other tools.
- The *Data* layer stores all the constraints and relational information in the LORCDB. The three tables used for indexing constraints and variables are also stored in this layer. In order to store the object in a relational database, Oracle™)9.*i* Database Management has been used, although it is also possible to use any database management that permits the object-relational paradigm. When a LORCDB is created or queried, the CORQL sentences are transformed into SQL sentences known by the database management system (Oracle in this case).

This architecture is modular so that those modules which evaluate the query can be modified or extended. For example, if it were possible to find another technique to work with trigonometric functions such as sine, cosine or tangent, then new modules would be added. Furthermore, if other better tools can be found to solve constraint problems, then the current tools can be easily adapted. This architecture permits the addition of new modules, due to the pattern strategy used, and it also allows modifications to provide more efficiency and versatility. This means that the architecture can be adapted to different types of constraints and the evaluation query modules can be extended.

In the following sections, some details about the submodules *search engine for the related constraints* and *Building Models depending on the Query* are presented. These submodules are in charge of analysing the query to reduce the computational complexity and to define an efficient plan for the evaluation of the query.

5. Search engine for related constraints

In order to obtain the evaluation of the projection operator over a constraint relation, two steps have to be carried out. The first step consists of deciding which part of the architecture can evaluate the query, and the second one consists of the detection of all the constraints related to the query depending on the first step. In this work, a computational improvement is proposed in order to obtain all the constraints related to the queries and optimize the query evaluation, by creating the different models only with those constraints necessary. This detection is possible thanks to the indexation of the variables and constraints in the relational database. Moreover, an algorithm is proposed to find sets of constraints that can be evaluated separately, thereby building smaller and easier problems to solve.

5.1. Step 1 – Selection of the solving tool and the type of model

Depending on the query, different parts of the architecture will be used. With the constraints related to the query, a *Model-Driven Query* is created. In Fig. 9, the decision tree shows the type of model created depending on the syntax of the query. There are three types of projection over variables to obtain: new constraints; values of the variables; or a minimum or maximum value of a variable. Depending on the syntax of the query, and the types of the related constraints, different models are created and different techniques are used. When the related constraints are of different types, the most general technique is used. For example, if the related constraints in a symbolic elimination of variables are polynomial and linear constraints, then the problem will be solved using cylindrical algebraic decomposition.

5.2. Step 2 – Identification of clusters of related constraints

Depending on the type of projection (numeric or symbolic), the constraints related to the projection are different. In order to understand this idea, graph theory is used, where each constraint of a CDB is represented as a node, and the variables represent the edges. Two constraints have an edge in common if they have a common variable.

For numeric resolution cases, a constraint c is a constraint related to the projection if and only if there exists a path from c to another constraint (node) with a variable of the projection.

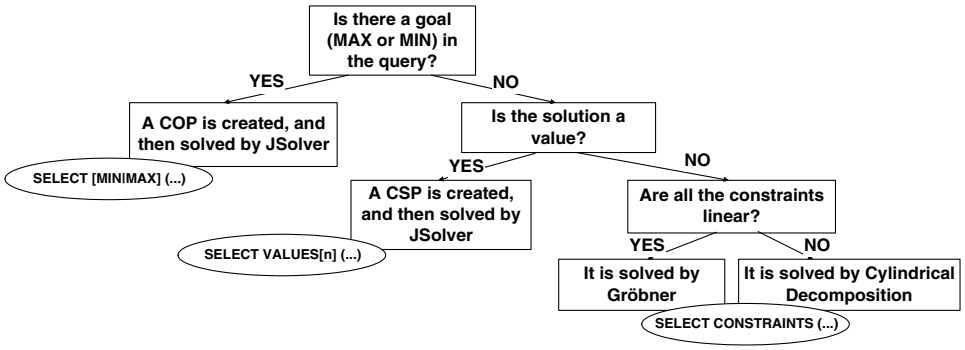

Fig. 9. Decision tree to choose what part of the architecture to use.

Definition 5.1 (*Cluster of related constraints for numeric projection (CRC)).* A cluster is a connected component of the defined graph, where the connected component has at least one variable of the projection.

For the example shown in Fig. 3 for the variables of the projection $\{a,b,c,d,e,f,g\}$ the cluster of related constraint is $\{A_1, A_2, M_1, M_2, M_3\}$.

For symbolic resolution cases, it is necessary to obtain new constraints inferred from those stored ones, and the variables that do not appear in the projection sentence are eliminated. In order to obtain the new constraints, Gröbner Bases and cylindrical algebraic decomposition are used, however, these techniques could be computationally very costly, hence in this step for symbolic resolution an algorithm is presented to reduce the use of these techniques in a significant way. In order to obtain the constraints involved in a query, the variables that appear in the projection (*VarQuerys*) are used, together with those variables that do not appear in the projection (*VarNoQuerys*) but that belong to the constraints necessary for the evaluation of the query. For example, if there are two constraints with the constraints $\{b = 2 * a; c = 3 * b\}$, and we want to obtain a new constraint with only the related variables a and c, then by combining these constraints it is possible to obtain a new constraint as $\{c = 6 * a\}$, because b is presented in both constraints and can therefore be substituted.

The objective in this step is to find all the clusters of related constraints where: if a *VarNoQuery* variable participates in a cluster, then this *VarNoQuery* variable appears in more than one constraint of the cluster so that elimination is possible, and new constraints with only *VarQuery* variables are obtained by using only a symbolic representation. This means that all the *VarNoQuery* variables could be replaced with *VarQuery* variables. In order to avoid duplicate solutions, these sets of constraints have to be minimal, which means that there is no subset of constraints that forms a cluster of related constraints. This idea generates a new definition.

Definition 5.2 (*Cluster of related constraints for symbolic projection (CRC)).* $G \equiv \bigcup_i \{c_i\}$ is a cluster of related constraints for a query $\mathbf{Q}$ (CRC($\mathbf{Q}$)) if

$$\forall c_i \in G | \text{VarNoQuerys}(\{c_i\}, \mathbf{Q}) \subseteq \text{VarNoQuerys}(G - c_i, \mathbf{Q})$$
$$\wedge\, \text{VarQuerys}(G, \mathbf{Q}) \neq \emptyset \wedge \nexists G' \subset G | G' \text{ is } (\text{CRC}(\mathbf{Q})),$$

where c_i is a constraint, VarNoQuerys(C,$\mathbf{Q}$) are the variables of the constraints C that do not appear in the query $\mathbf{Q}$, and VarQuerys(C,$\mathbf{Q}$) are the variables of the constraints C that appear in the query $\mathbf{Q}$.

For the example shown in Fig. 3 for the variables of the projection a,b,c,d,e,f,g, the clusters of related constraints are $\{A_1, M_1, M_2\}, \{A_2, M_2, M_3\}$ and $\{A_1, A_2, M_1, M_3\}$.

First of all, we present an example to explain the idea of finding the related constraints in a query. For example, if a query asks about the variables $\{a, b, e\}$, where the constraints are $\{a + b = c, c + d = e\}$, then these constraints do not form a cluster of related constraints since variable d is not a *VarQuery* nor can it be substituted by any of the *VarQuery* variables $\{a, b, e\}$. This means that a CRC is a set of constraints whose variables are in the query or are in other constraints of the CRC, and thereby they can be substituted by *VarQuery* variables. However, if the previous example has the variables $\{a, b, e, d\}$ as *VarQuery* variables, then these constraints form a CRC.

However, another definition is necessary to describe the algorithm to obtain the clusters of related constraints:

Definition 5.3. Unsolved constraintThis is a constraint with one or more than one *VarNoQuery* variable that is only in this constraint, or in another *unsolved constraint*. This means that this *varNoQuery* variable cannot be substituted by another.

Since the constraints related to the query by the variables are unknown, it is necessary to find the CRCs by extending the relation between constraints through the proposed indexation of constraint objects. Algorithm 1 expands the search, starting with the constraints with *VarQuery* variables, which are the constraint-variable attributes that appear in the projection. The algorithm uses a list of constraints in the search (*ConstraintList*):

Algorithm 1. Obtaining CRCs for Symbolic Projection

```
List ConstraintList := new List( )
List CRC := new List( )
Boolean end := LoadConstraintList(ConstraintList)
while (¬end) do
    deleteUnSolvedConstraints(ConstraintList)
    CRC.add(obtainCRC(ConstraintList))
    end := LoadConstraintList(ConstraintList)
```

Description of methods:

- *Boolean LoadConstraintList(List):* The first time this method is called (when *ConstraintList* is empty), the method obtains the constraints related to the *VarQuerys*. The rest of the times, the method obtains the constraints related to the VarNoQuerys of the *ConstraintList* which have not already been included. All the information is stored in *ConstraintList*, but if *ConstraintList* does not change, the method returns *false* to indicate that the search has finished.

- *List obtainCRC(List):* This method identifies if there is a set of constraints that satisfies the CRC definition, and returns it. In this part of the algorithm the graph theory is also used, where the constraints are the nodes and the *VarNoQuery* variables are the edges. In order to determine if a set of constraints forms one or more CRCs, the method looks for the connected components. These connected components form a CRC if all the *VarNoQuery* variables are in at least two constraints and it is minimal. In order to determine if the CRC is minimal, it is necessary to analyse only the *VarNoQuery* variables that appear in more than two constraints. If a connected component cc_i has a constraint c with a *VarNoQuery* variable that appear in more than two constraints, then the method analyses whether $\{cc_i - c\}$ is a CRC: if $\{cc_i - c\}$ is not a CRC, it means that cc_i is a CRC. Otherwise, the process continues to determine whether $\{cc_i - c\}$ is a minimal CRC.
  When a CRC is determined in the *ConstraintList*, these constraints are deleted from *ConstraintList*.
- *void deleteUnSolvedConstraints(List):* This method deletes the unsolved constraint according to Definition 5.3.

The algorithm to obtain the CRCs is better explained with a more complex example, as shown in Fig. 10 where the *VarQuery* variables are $\{a, b, c, d, u, a1, b1, c1, d1, y1, a2\}$. The following syntax of projection is used:

```
SELECT CONSTRAINTS (Behaviour.a, Behaviour.b, Behaviour.c,
   Component.Behaviour.d, Behaviour.u, Behaviour.al, Behaviour.bl,
   Behaviour.cl, Behaviour.dl, Behaviour.yl, Behaviour.a2)
FROM Component
```

For the example in Fig. 10 and the variables of the last query, the names of the components related to the constraints obtained with the *LoadConstraintList* method are:

```
{A1, A2, M3, A7, A23, S1, A11, M4, A18}
```

In fact the algorithm works with the field *idConstraint*, although here the names of the components are used in order to clarify the explanation. The known constraints are those shown in Step *a* of Fig. 11, obtained with the method *LoadConstraintList*. After obtaining these constraints, the algorithm checks if there are any CRCs among them (*obtainCRC* method). In the example, there are no CRCs, thereby the process continues to look for more related constraints. The next search will be executed with the variables of the obtained constraints that do not appear in the query (*VarNoQuery* variables). In this case, the search uses only the variables $\{e, f, g, s, k, m1, n1, z1, x1, h2\}$, and it adds the new constraints, related to these variables, to the previous list. After this execution the *ConstraintList* has the constraints:

```
{A1, A2, M3, A7, A9, S1, A11, M4, A18, M1, A3, A8, A23, M6, A10, M9}
```

The algorithm searches among the constraints (shown in Step *b* of Fig. 11) to find the CRCs. In this case the algorithm finds two CRCs:

```
{A1, A2, M3, M1}
```
and
```
{A9, S1, A11, M11, M6, A10}
```

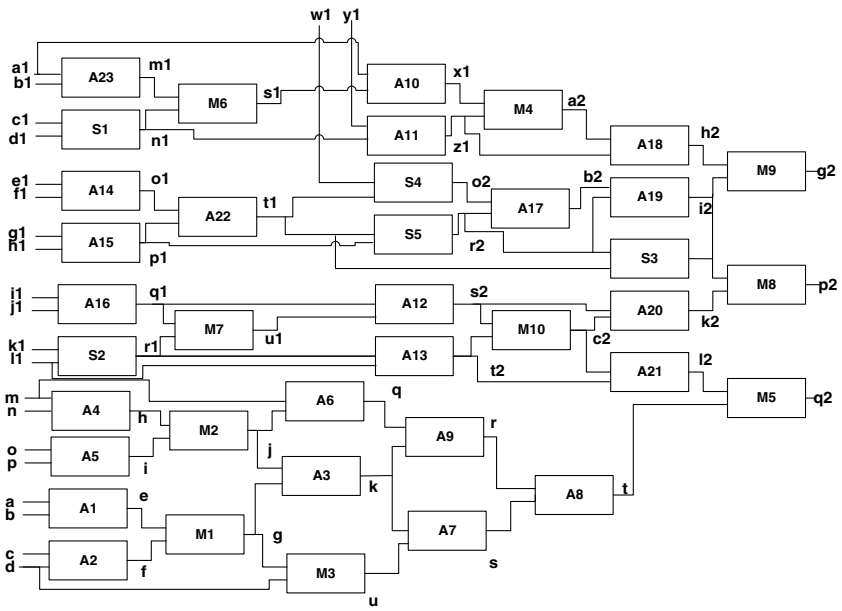

Fig. 10. Example for projection.

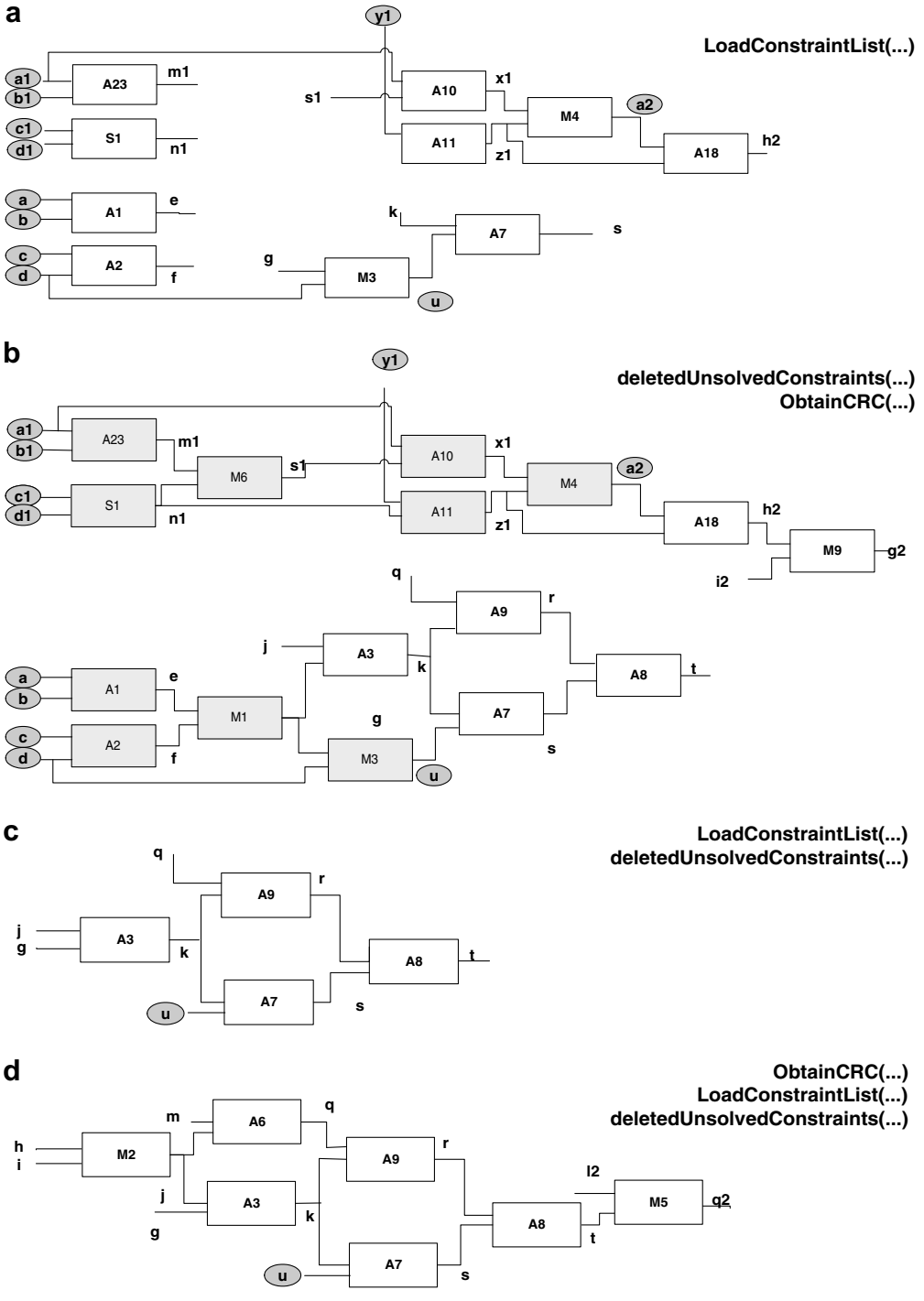

Fig. 11. Example of the steps for the CRC algorithm.

Hence the rest of the constraints of *ConstraintList* are:

{A7, A3, M4, A8, A18, M9}

However, some constraints of this list never participate in a CRC because they have at least one variable that is not in the query and is not in another constraint, thereby they are never substituted by *VarQuery* variables. An example of these Unsolved Constraints is M9, since g2 is not a *VarQuery*, and this variable does not appear in other constraints and therefore, it is not substituted by *VarQuery* variables. This also means that A18 is an unsolved constraint since it has a variable ($h2$) which is

only in an unsolved constraint. After these eliminations performed by the method *deleteUnSolvedConstraints*, the list (shown in Step *c* of Fig. 11) becomes:

```
{A7, A3, M4, A8}
```

The search continues with the variables $\{j,q,t\}$, and the new list (shown in Step *d* of Fig. 11) becomes:

```
{A7, A3, M4, A8, M2, A6, M5}
```

In this case M5 is an unsolved constraint because it has the variable $q2$ and this variable is only in this constraint and is not a VarQuery. Therefore, A8 is also an unsolved constraint because the variable t is only in an unsolved constraint (M5). As a consequence, M4, A7, A3, A6 and M2 are also unsolved constraints because they have variables in unsolved constraints. At this point the process finishes, since there are no constraints left to analyse in *ConstraintList*.

It is very interesting to note that these CRCs can be solved in a parallel or distributed way, owing to the independence of their variables. This characteristic permits the computation time to be improved due to the added efficiency of the solution, both for numeric problems and for symbolic problems. If no CRC is found for a query, the evaluation of the query does not return any information.

The computational complexity of the algorithm is related to the number of constraints stored in the LORCDB and not to the number of variables. It is also related to the distance between the variables of the query, hence the following definition is required.

Definition 5.4 (*Distance between variables).* Representing the constraints of a relation as a graph, where the nodes are the constraints, there is an edge between two nodes if they have the same variable. If C_1 and C_2 are constraints, a and b variables, and $a \in C_1$ and $b \in C_2$, then the distance between the two variables (Distance(a,b)) is the minimal distance between C_1 and C_2. The distance between two variables of the same constraint is 0.

In order to analyse the number of steps needed to extend the search for constraints and obtain the CRCs for a query, it is necessary to study two scenarios, which are:

- When it is possible to find a CRC for the *VarQuery* variables defined for the projection. In each step of the algorithm, the search for the constraints is expanded depending on both *VarQuery* variables and *VarNoQuery* variables included in previous expansions. Hence, the worst case is half of the longest distance between two variables of the same CRC. For the example of Fig. 11, the distance between the variables $a1$ and $a2$ is 4, and both of them are *VarQuery* variables, which means that it is necessary to execute the search algorithm for constraints twice.
- When it is not possible to find a CRC for the *VarQuery* variables defined for the projection. It is not always possible to find CRCs for any constraints, since the constraints studied in the algorithm cannot participate in those CRCs which cannot be solved (such as M5, A8, M4, ...) because they are unsolved constraints. Another unfavourable case is the greatest distance between a *VarQuery* and another *VarNoQuery* which is impossible to solve because this variable belongs to an *unsolved constraint*. For the example of Fig. 11, the distance between the variables u and $q2$ is 3, where u is a *VarQuery* but $q2$ is a *VarNoQuery*, which means that it is necessary to execute the search for constraints three times until an unsolved constraint is found.

When the set of constraints is found, it is necessary to detect if there is a CRC. This occurs when there is a connected component and all the *VarNoQuery* variables are at least in two constraints. By using the graph theory, the complexity to determine if a graph is connected is $O(n)$, where n is the number of constraints. In order to know if all the variables are in two constraints, the complexity is $O(v)$, where v is the number of *VarNoQuery*. Therefore, the computational complexity of the algorithm to obtain CRCs is:

$$O\left(Max\left(\frac{\max\limits_{\forall a,b}(Distance(a,b))}{2}, \max\limits_{\forall c,d}(\text{Distance}(c,d))\right) * (n+v)\right),$$

where a, b, c are VarQuerys and d is a variable of an unsolved constraint.

6. Building models depending on the type of query

In this section, the creation of the different models for the projections are presented, using the CRC algorithm explained in the section above. Depending on the query, one of the aforementioned techniques is used for its evaluation.

6.1. Symbolic model for equality constraint query

This type of model is created when a new set of constraints has to be obtained using those stored, and the stored constraints are equations. In this case, the variables which do not appear in the query but are related to constraints that contain the variables of the query, must be replaced by variables of the query.

An example of a query solved using this symbolic technique is:

```
SELECT CONSTRAINTS (Behaviour.m, Behaviour.j, Behaviour.g,
   Behaviour.d, Behaviour.r, Behaviour.s) FROM Component
```

This type of query is used in diagnosis in order to know the CARCs (Definition 3.3) of a system if the sensors are located in the variables $\{m,j,g,d,r,s\}$.

For these variables and using the explained algorithm, three different CRCs are found, as shown in Fig. 12. This figure shows an example formed by equality polynomial constraints and the new constraints inferred from the originals. The different lines outline the three sets of constraints, and to the right of the figure, the new constraints related with each set of constraints are presented.

Since all the constraints are labelled as polynomial equality constraints, Gröbner Bases technique implemented in Mathematica™) *v*.5 is used. The prototype of the function is:

```
GroebnerBasis[{CRC}, {VarQuerys}, {VarNoQuerys}]
```

For the system presented in the figure above, an example of the model for equality constraint projection is:

```
GroebnerBasis[{g*d-u=0, m+j-q=0, q*k-r=0, k+u-s=0},
   {m,j,g,d,r,s}, {k,q,u}]
```

whose result is: $\{d*g*j+d*g*m+r-j*s-m*s=0\}$.

It is also possible to combine classic and constraint attributes using projection. For example, to know the CARCs of the systems and the Context Sets for these CARCs, the query could be:

```
SELECT Name, CONSTRAINTS (Behaviour.m, Behaviour.j, Behaviour.g,
   Behaviour.d, Behaviour.r, Behaviour.s) FROM Component
```

The result is shown in Table 2, where three different CARCs appear in the column *Behaviour*, and the components related with them in the column *Name*.

6.2. Symbolic model for inequality constraint query

When a CRC of a query has at least one inequality constraint, it is necessary to use cylindrical algebraic decomposition. If, for example, the system of Fig. 10 has constraints associated to the components such as:

```
A16: il+jl ⩽ ql
S2: kl-ll ⩽ rl
M7: ql*rl ⩽ ul
A12: ql+ul ⩽ s2
```

An example of a query in fault diagnosis could be: how does the system work if the sensors are located in the variables $\{i1,j1,k1,l1,s2\}$? The query is:

```
SELECT CONSTRAINTS (Behaviour.il, Behaviour.jl,
   Behaviour.kl, Behaviour.ll, Behaviour.s2) FROM Component
```

Since the constraints are inequalities, the *cylindrical algebraic decomposition* module is used. In this case Mathematica™) v.5 is also used. The syntax is:

```
Reduce[Exists[{VarNoQuery},CRC],{VarQuery}]
```

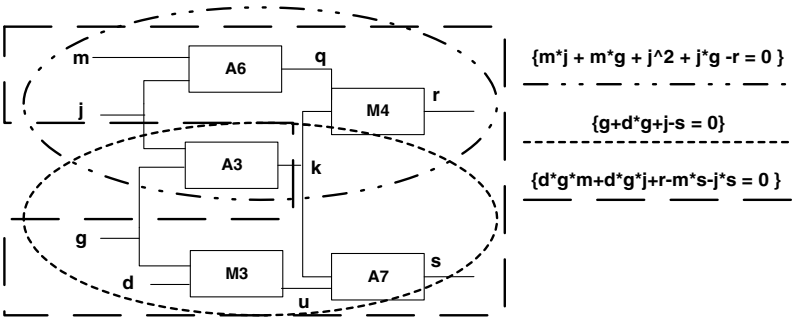

Fig. 12. CRCs detected by the query.

Table 2
Classic and constraints attributes (context sets and CARCs)

Name	Behaviour(j,m,g,d,r,s)
M3	$\{-g-d*g-j+s=0\}$
A3	$\{-g-d*g-j+s=0\}$
A7	$\{-g-d*g-j+s=0\}$
A6	$\{-(g*j)-j-g*m-j*m+r^2=0\}$
A3	$\{-(g*j)-j-g*m-j*m+r^2=0\}$
M4	$\{-(g*j)-j-g*m-j*m+r^2=0\}$
M3	$\{d*g*j+d*g*m+r-j*s-m*s=0\}$
A6	$\{d*g*j+d*g*m+r-j*s-m*s=0\}$
M4	$\{d*g*j+d*g*m+r-j*s-m*s=0\}$
A7	$\{d*g*j+d*g*m+r-j*s-m*s=0\}$

For the example, the model created to solve the projection is:

```
Reduce[Exists[{ql, rl, ul},il + jl ⩽ ql ∧ ql * rl⩽ul ∧ kl - ll⩽rl ∧ ql + ul⩽s2],{il, jl, kl, ll, s2}]
```

In this case, the output of the function is:

```
jl < -il ∨ jl = -il ∧ (ll ⩾ 1 + kl ∧ s2 ⩾ 0 ∨ ll > 1 + kl) ∨ jl > -il ∧ (ll < 1 + kl ∧ s2 ⩾ il + jl + il kl + jl kl -
il ll - jl ll ∨ ll = 1 + kl ∧ s2 ⩾ 0 ∨ ll > 1 + kl)
```

However, a *constraint attribute* cannot be formed by $\vee$ operators. It is thereby necessary to represent it in disjunctive normal form in order to present the solution as a set of tuples, as presented in Table 3. The elimination of variables sometimes obtains non-polynomial constraints as a result, and in this case the output of the query evaluation is empty to maintain the closed property.

6.3. Numeric model for constraint satisfaction problems

When the answer of a query is not a new constraint but is a value or a set of values, then it is not necessary to use symbolic techniques. This type of query is used to obtain an extensional representation of constraint-variable attributes. An example in fault diagnosis is when the user wants to know the values of some variables with sensors. An example of numeric projection can be the search for 10 possible tuples of values for the variables `g2`, `p2`, `q2`:

```
SELECT VALUES[10](Behaviour.g2, Behaviour.p2, Behaviour.q2)
    FROM Component
```

It is also possible to instantiate some variables, for example the inputs of the system, in order to know some possible outputs, when the user knows the values of some variables but wants to know the values of others. For instance, what is the value of u if the value of the sensors are: $\{a=1,b=3,c=2,d=1\}$? The query for this example is:

```
SELECT VALUES[1](Behaviour.u) FROM Component
WHERE Behaviour.a=1 AND Behaviour.b=3
    AND Behaviour.c=2 AND Behaviour.d=1
```

In order to know the values of variables, a CSP is created dynamically by using the constraints obtained from the database, and if it is necessary, by including the values of the variables of the query. In the CSP all the variables are instantiated although only *VarQuery* variables are presented as the solution. A common issue in diagnosis is the problem of knowing whether the system works correctly for an observational model. This is discovered by carrying out the projection over a constraint attribute. An example of this type of query can be: Does the system work correctly for the OM $\{a=1,b=3,c=2,d=1,u=15\}$? This query is:

Table 3
Output from projection over inequality constraints

Behaviour($i1,j1,k1,l1,s2$)
$j1 < -i1$
$j1 = -i1 \wedge (l1 \geqslant 1+k1 \wedge s2 \geqslant 0 \vee l1 < 1+k1)$
$j1 > -i1 \wedge l1 < 1+k1 \wedge s2 \geqslant i1+j1+i1\ k1+j1\ k1-i1\ l1-j1\ l1$
$j1 > -i1 \wedge l1 = 1+k1 \wedge s2 \geqslant 0$
$j1 > -i1 \wedge \vee l1 > 1+k1$

```
SELECT VALUES[1] (Behaviour.a, Behaviour.b, Behaviour.c,
   Behaviour.d, Behaviour.u) FROM Component
WHERE Behaviour.a=1 AND Behaviour.b=3 AND Behaviour.c=2
   AND Behaviour.d=1 AND Behaviour.u=15
```

The problem involves constraint satisfiability in CDBs. If the result is not empty, it means that the involved constraints are satisfiable for the instantiation of the variables in the query. In order to know the solution, a CSP is created with the related constraints with the *VarQuery* variables *a*, *b*, *c*, *d* and *u*. For the example no solution is found, which means that the system is not working correctly.

6.4. Numeric model for constraint optimization problems

When a numeric evaluation obtains several tuples and the user wants to select from among the possible options, a COP has to be created. CORQL makes it possible to obtain the maximum or minimum value of a variable, and an example of a query that creates this type of module is:

```
SELECT MIN VALUE(Behaviour.u) FROM Component
```

A COP is built according to a query in a similar way to a CSP, and it is also possible to add new constraints to the COP defined in the query. An example can be:

```
SELECT MIN VALUE(Behaviour.u) FROM Component
WHERE Behaviour.a>=3 AND Behaviour.b>=1
   AND Behaviour.b< 5 AND Behaviour.c>=2
   AND Behaviour.d<=3 AND Behaviour.d>=0
```

With the constraints of the attribute *Behaviour* of the relation *Components* and the added constraints in the query, a COP is created in order to infer the value of the variable, *u* in this example. This COP is created dynamically by using the objective of the query, the constraints obtained from the CDB, *VarNoQuery* and the instantiated variables in the query. Fig. 13 shows the COP built automatically for the example, solved using JSolver™) [41].

7. Experimental results for the projection operator

In this section, all the types of projections are analysed in order to show how the design of the architecture affects the efficiency of the query evaluation. For this reason, several comparisons are carried out between the time of evaluation with and without using CRCs for symbolic projection, and also with and without using heuristics for numeric projection.

All the measures have been obtained using an AMD Athlon™) 64 Dual Core Processor 4200 + 2.21 GHz, where the Database Management System Oracle is running. All the measures are presented in milliseconds.

All solutions are presented over float domain problems, since natural or integer domains cannot always be used for symbolic solvers [33,26]. The results shown for CSPs are defined over the float domain because this type of problem is more complex to solve than over the integer domain, and hence the worst domain case for the numeric projection is analysed in this work.

The problems used in the tests are generated randomly, whereby different parameters are changed. All the constraints have the form {v_1 operation v_2 comparator v_3}. Each constraint of a problem has a set of variables (v_1, v_2, v_3), the

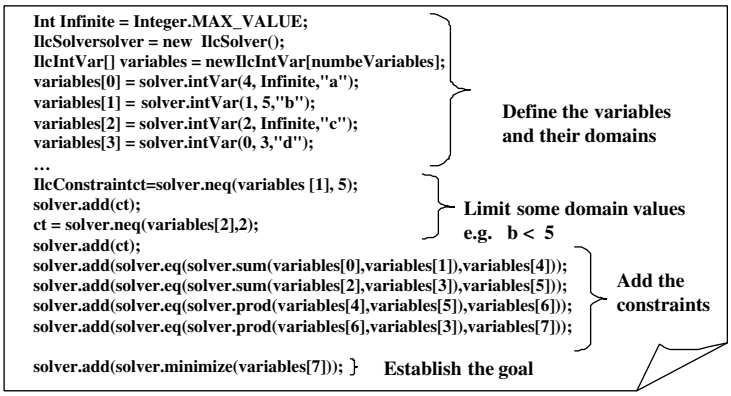

Fig. 13. Example of model for a COP.

`operations` (+, *, −) and the `comparators` (=, ⩽, <, ⩾, >). In this case, these different parameters are fixed to build problems that can be diagnosed. The relation between the constraints will be defined for those variables shared between the different constraints.

The random problems are created incrementally, by adding new constraints whose variables can be new variables or variables already included in the problem. The systems created in this way are composed of one connected component according to the definition of graph introduced in Section 5.2. For the different types of projection, the variables that will be obtained (in a numeric or symbolic way) are those variables whose distance is the greatest (according to Definition 5.4), which implies the worst case of the projection for a constraint problem.

7.1. Symbolic projection evaluation

Symbolic projection allows new constraints to be obtained with the same solutions as the original constraints. In order to obtain these new constraints it is necessary to analyse the clusters of related constraints for both equality and inequality constraints.

Thanks to the indexation between variables and constraints, it is possible to use the developed algorithm in order to obtain the CRCs. In this section, a comparison between the run times is made to obtain the projection with different examples. To show how the evaluation time decreases when only CRCs are analysed, two options are compared. The first option is to study all the possibilities, whereas the second one is our proposal to improve the computational time. Both options obtain the same results, since the CRCs evaluation is a subset of all the possibilities where the sets of constraints without solutions or with duplications are not analysed. The compared options are:

- *Checking all the combinations:* This involves the analysis of $(2^n - 1)$ combinations. With these $(2^n - 1)$ sets of constraints, either Gröbner Bases or cylindrical decomposition is used to obtain new constraints with only *VarQuery* variables depending on the type of related constraints.
- *Obtaining the CRCs.* The designed architecture allows the definition of the relation between constraints and variables, and an algorithm has been developed in order to obtain the sets of constraints related to a projection in accordance with their variables. This means that symbolic-elimination techniques can be only used with some sets of constraints.

7.1.1. Test queries for symbolic projection performance

The test queries for symbolic elimination are divided into two different groups, since the execution time will be very different depending on the technique used to evaluate the projection (Gröbner Bases or cylindrical decomposition). In both cases, different random examples with a different number of constraints have been created, thereby obtaining different evaluation times.

First the Gröbner Bases test queries are presented, which means that all the constraints of the system have to be equality constraints. In Fig. 14 (presented in logarithm scale), the mean times using 10 instances for the evaluation of different queries over systems with a different number of constraints are shown. This figure presents the different times both with the proposed improvement using CRCs, and without this improvement. The evaluation time used to obtain the CRCs is also shown as a part of the whole time necessary to evaluate the whole query. In Fig. 14, it is also possible to observe how the evaluation time for the solution without improvement increases exponentially according to the number of options, while the use of CRCs drastically reduces the evaluation time. Using more than 18 constraints, the evaluation times without improvement go sky high. In the bar diagram this is presented with the value 10^9, although the processes take longer than five days. This behaviour is expected since the number of variables increases when the number of constraints increases, and the evaluation time is exponential according to the number of variables, as was explained in Section 4.2.1.

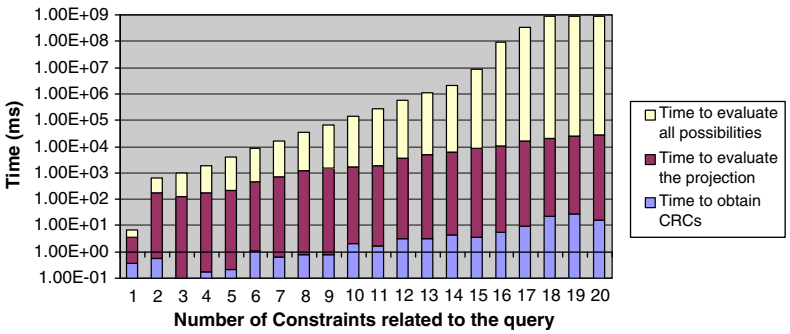

Fig. 14. Runtime for Gröbner symbolic projection.

Obviously, the evaluation time depends on the structure of the constraint relation (relation between constraints), the number of constraints, the number of variables of the projection and the type of constraints (linear or polynomial). Fig. 15 shows, by using five different examples, how the evaluation time changes for a system when the number of *VarQuery* variables changes and the type of constraints changes.

The results presented in Fig. 15 have been generated by a randomly created static structure whilst changing the number of variables of the projection and the number of polynomial constraints for different tests. In Fig. 15, it is possible to show how the time drops when the number of variables of the projection grows because the distance between the variables are smaller and the CRCs are easier to find and solve. If the number of polynomial constraints changes, the results do not permit a relation to be found between the number of polynomial constraints and the execution time when Gröbner Bases are used, since the algorithm complexity is exponential according to the number of variables that have to be eliminated.

In the cylindrical algebraic decomposition case, it is possible to have inequality constraints in the systems. In Fig. 16, the mean times using 10 instances for the evaluation of different queries over the system with a different number of constraints are presented. The comparison is defined both with the proposed improvement, and without this improvement. The part of the evaluation time used to obtain the CRCs is also shown. In Fig. 16, it is possible to observe how the evaluation time for the solution without improvement increases exponentially, according to the number of constraints, while the use of CRCs drastically reduces the evaluation time. Using more than 14 constraints, the evaluation times without improvement go sky high. In the bar diagram this is presented with the value 10^9, although the processes take longer than 8 days.

For cylindrical decomposition, the evaluation time is double exponential according to the number of cells, and in a similar way to Gröbner Bases, the evaluation time depends on the structure of the system, the number of constraints and the number of variables of the projection. However, in this is case, the type of constraint (linear or polynomial) is also important, since polynomial constraints help to generate more cells than do linear constraints. Fig. 17 shows how the evaluation time changes for a constraint relation when the number of *VarQuery* variables changes and the type of constraints changes.

The results presented in Fig. 17 have been generated by a randomly created static structure whilst changing the number of variables of the projection and the number of polynomial constraints for different tests. Different tests have been executed, analysing the time when the number of variables of the projection and the number of polynomial constraints change. In Fig. 17, it is possible to show how, by using five different examples, the time drops when the number of variables of the projection grows, for the same reason as was given in Gröbner Bases. In the case where the number of polynomial constraints is changed, it is possible to observe how the time increases when the number of polynomial constraints grows.

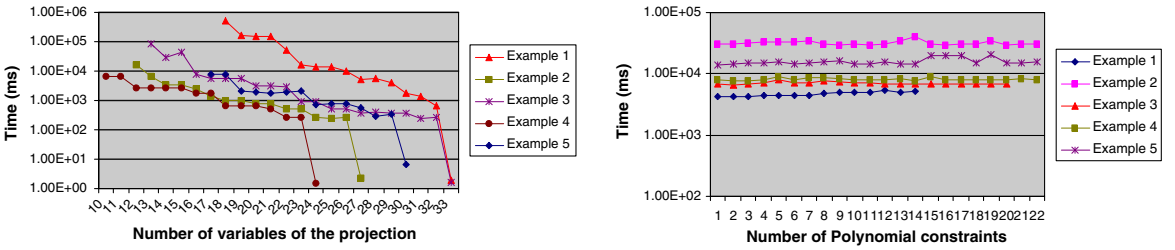

Fig. 15. Runtime while changing parameters for Gröbner symbolic projection.

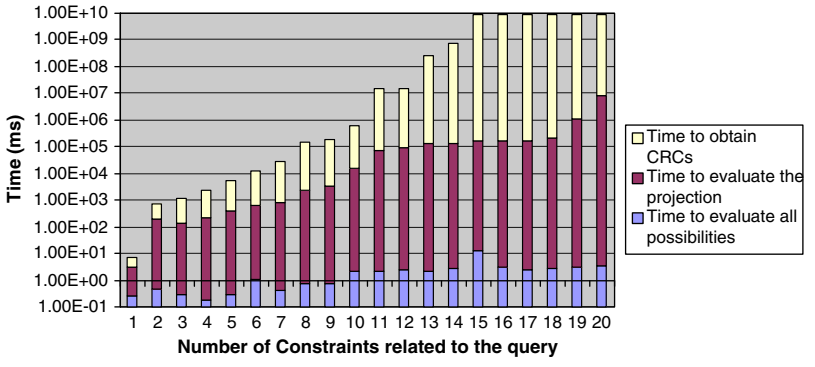

Fig. 16. Runtime for cylindrical decomposition symbolic projection.

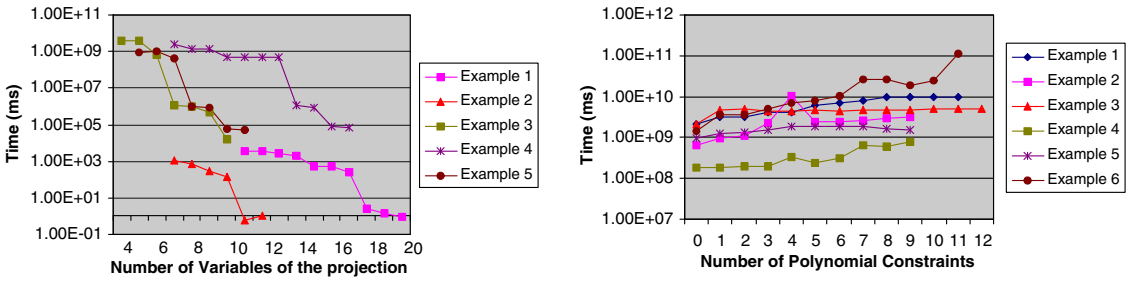

Fig. 17. Runtime while changing parameters for cylindrical decomposition symbolic projection.

7.2. Numeric projection evaluation

Some of the different heuristics that can be used in order to solve CSPs are presented in this paper. It is well known that the order in which variables and values are selected has a dramatic effect on the algorithm efficiency [18]. For this reason, our architecture makes using these heuristics easier in order to solve the models, and some comparisons are presented in this work:

- *No heuristics:* This means solving CSPs with no heuristics. In this case, the variables of constraints are selected in the order obtained from the CDB.
- *By domain:* This means starting with the variable with the smallest domain. The most used variable-ordering heuristic selects the variable with the minimum number of values in its current domain [32]. This is possible since the LORCDB stores the minimum and maximum value of each variable as a box consistency.
- *By type of constraint:* This means starting with the variables that belong to the linear constraint scope. The labels of the constraints are also stored, and it is thereby possible to analyse the variables of linear constraints first and the variables of polynomial constraints after.

7.2.1. Test queries with numeric projection

In this section, some examples of queries which obtain instantiated values of variables are presented. A set of random examples and projections have been executed, whereby some parameters have been changed such as the number of constraints and the number of polynomial constraints.

Some measurement times for the evaluation of each query using the different heuristics are presented in Fig. 18. These heuristics solve the same types of problems but the number of constraints is changed. As shown in this diagram, the execution time depends on the number of constraints. Fig. 18 also shows how the evaluation time for the same problem changes depending on the heuristic used. In this figure it is possible to observe the importance of selecting a good heuristic [19], and how the evaluation time can consequently increase or decrease. For the random examples for 14 and 19 constraints, the difference is very high. Furthermore, whether the type of constraint is always important has been analysed. In Fig. 19, the evaluation time is presented for five different examples where the structure is always the same for each example, but where the linear constraints are changed into polynomial constraints. This means that the behaviour of the constraints changes although the structure is maintained. All the tests have been generated by a randomly created static structure whilst the type of constraint is changed. For the solved examples, the execution time experiences a slight decrease when the number of

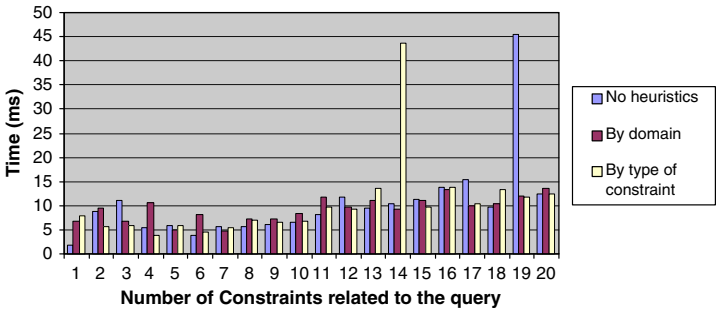

Fig. 18. Runtime for numeric projection.

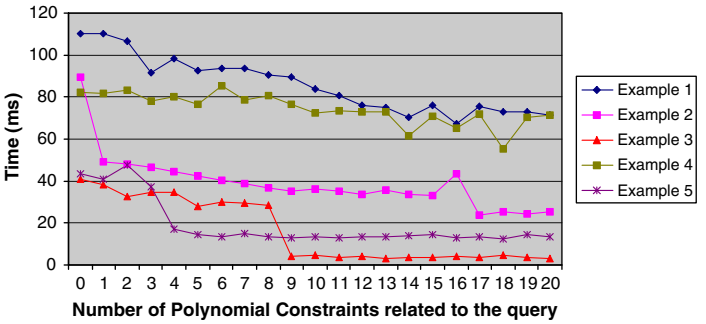

Fig. 19. Runtime while changing type of constraints.

polynomial constraints increases. This is due to the type of problems created, and the reduction of the possible values in the domain of the solution through the use of polynomial constraints.

In order to solve a CSP, all the variables of the constraints have to be instantiated. This means that for a numeric projection all the variables have to be instantiated, although they have not been presented as a solution of the projection. For this reason, the number of variables of the projection has not been analysed for numeric resolution purposes.

7.3. Projection over the maximum or minimum value of a constraint-variable attribute

When a projection over a set of constraints requires the minimum or maximum value of a variable, it is necessary to build a constraint optimization problem with all the constraints which belong to the CRCs. By using the proposed architecture which stores the box consistency of each variable, it is possible to use heuristics to make the evaluation more efficient. In this case two ways are compared to show how the evaluation time drops sharply when using the box consistency. The different ways that are compared are:

- *Analysing all the solutions:* Building a COP where the variable of the projection is defined as the objective of the COP and all the solutions are analysed to find the best value from among all the solutions found.
- *Starting the search by instantiating the objective variable:* As explained in the previous section, the selection of a good heuristics to solve a CSP is an important factor for the improvement of the evaluation time. In this case, we propose the use of value-ordering heuristics which define the order of values to instantiate the variables to find the different solutions of the COP. Generally, value-ordering heuristics suppose that good values are those which are more likely to participate in solutions [19]. Another possibility is sorting the value of the variable by increasing the size of the resulting domain [25]. Our idea to solve a COP is to avoid studying all the possible solutions to obtain the maximum or minimum value of a constraint-variable attribute, trying to know if a tuple of values where the objective value is the best option is a solution. In this case, it is not necessary to continue looking for another solution: the solution found is the best solution. If the tuple is not a solution another solution can be sought although the objective will be slightly worse. This implies that to find the first solution is to find the best solution because determine the value-ordering of the objective variable is determined. To this end, we propose starting the instantiation for the objective variable with a value-ordering by using the most promising value. If the query evaluation consists of obtaining the minimum value of a variable, the search starts with the minimum value of the objective variable stored in the LORCDB or with the maximum value otherwise. Hence, we transform the analysis of all solutions into the search for the first solution. For example, if r has to be maximised and its box consistency is defined over 10–20, the heuristics starts looking for a solution where $r = 20$. If this solution is not found the process continues until the first solution is found and the process finishes. If this heuristics is not used, and for example a solution is found for $r = 15$, the process has to proceed until the whole domain is analysed.

7.3.1. Test queries over maximum value

In this case, different random queries are also evaluated by increasing the number of constraints of the COP to obtain the maximum value of a variable. In Fig. 20, the different measurements are presented which are produced by changing the number of constraints. The comparison is defined between the analysis of all solutions and the resolution of a CSP by starting with the objective variable and the appropriate instantiated value.

In this figure, it is possible to observe how the computational time is drastically reduced by avoiding the study of all the solutions. This heuristic transforms the study of all solutions into the search of a solution with a defined order of the variables and the instantiation values. In the examples, the computation time is improved 1000 or even 10,000 times in comparison with all other solutions.

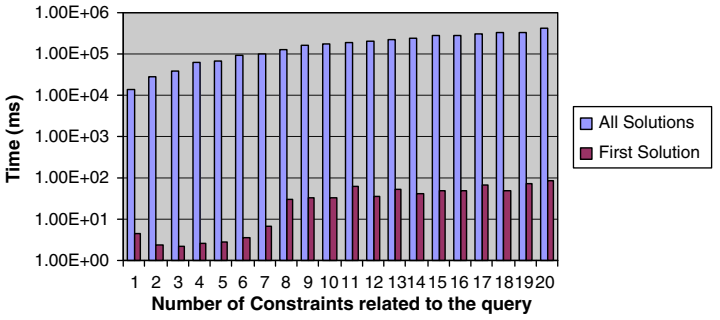

Fig. 20. Runtime for projection over maximum value.

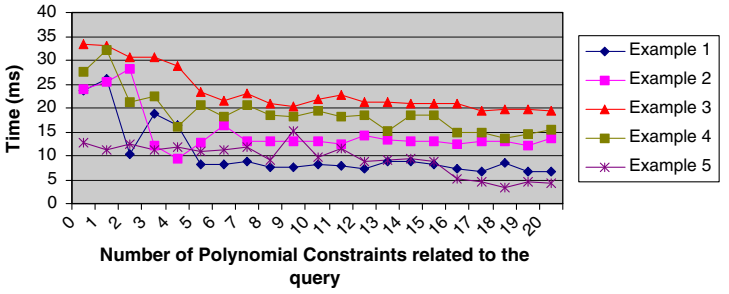

Fig. 21. Runtime for projection over maximum value while changing the type of constraint.

In this case, other tests have also been executed which are produced by changing the number of polynomial constraints for the same structure. The execution time for different constraint relations, represented with different symbols, are presented in Fig. 21. In this case, a relation between the number of polynomial constraints and evaluation time exists. This is due to the type of problems created, and the reduction of the possible values in the domain of the solution through the use of polynomial constraints.

8. Conclusions and future work

This work presents a new architecture for CDB in order to store constraint information as a new data type. For querying over the CDB, an extension of SQL called CORQL has been provided. In this architecture, the constraints are labelled and stored as objects in a object-relational database in order to create different models and select the most appropriate technique and tool to evaluate each query. Furthermore, important optimization improvements are presented, such as:

- The indexation of variables in order to improve the computation time of the query or the use of heuristics to solve CSP.
- An algorithm to obtain the CRCs, which improves the computation time in symbolic techniques. This algorithm creates constraint problems with only those constraints related to the query.
- The storing of the box consistency to reduce the options to solve the CSPs and COPs.
- The labelling of the constraints to decide which tool is more appropriate for the evaluation of a query where the constraints are involved.

The architecture is designed in layers which provides a means of interaction between the user and the system. The use of patterns allows changes and extensions to the different modules and tools, proving itself to be flexible and versatile. The architecture also offers a way of including constraints into the query which are not stored in the CDB.

Finally, some examples and computation times are presented in order to demonstrate how the architecture helps in the evaluation of queries with the projection operator.

As future work, we suggest an extension to the selection, Cartesian product, union and difference operators in order to complete the relational algebra for *Constraint* type. As for SQL sentences, it is necessary to offer all the possibilities of standard SQL, such as UPDATE, REMOVE and other aggregation functions.

Acknowledgements

This work has been partially funded by the Ministry of Science and Technology of Spain (DPI2006-15476-C02-00) and the European Regional Development Fund (ERDF/FEDER).

References

[1] S. Basu, Algorithms in semi-algebraic geometry, Ph.D. Thesis, 1996.
[2] S. Basu, R. Dhandapani, R. Pollack, On the realizable weaving patterns of polynomial curves in r_3, in: Graph Drawing, Lecture Notes in Computer Science, vol. 3383, Springer, 2004.
[3] F. Benhamou, L. Granvilliers, Continuous and interval constraints, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Foundations of Artificial Intelligence, Elsevier Science Publishers, Amsterdam, The Netherlands, 2006 (Chapter 16).
[4] R.E. Bixby, E.A. Boyd, R.Z. Ríos-Mercado (Eds.), Integer Programming and Combinatorial Optimization, Sixth International IPCO Conference, Houston, Texas, USA, June 22–24, Proceedings, Lecture Notes in Computer Science, vol. 1412, Springer, 1998.
[5] A. Brodsky, V.E. Segal, J. Chen, P.A. Exarkhopoulo, The CCUBE constraint object-oriented database system, in: ACM SIGMOD Conference, ACM Press, 1999.
[6] C.W. Brown, Improved projection for cylindrical algebraic decomposition, Journal of Symbolic Computation 32 (5) (2001) 447–465.
[7] B. Buchberger, Gröbner bases: an algorithmic method in polynomial ideal theory, in: N.K. Bose (Ed.), Multidimensional Systems Theory, 1985, pp. 184–232.
[8] J.H. Byon, P.Z. Revesz, DISCO: a constraint database system with sets, in: Constraint DataBases, Lecture Notes in Computer Science, vol. 1034, Springer, 1995.
[9] S. Ceri, G. Gottlob, L. Tanca, What you always wanted to know about Datalog (and never dared to ask), IEEE Transactions on Knowledge and Data Engineering 1 (1) (1989) 146–166.
[10] J. Chomicki, D.Q. Goldin, G.M. Kuper, D. Toman, Variable independence in constraint databases, IEEE Transactions on Knowledge and Data Engineering 15 (6) (2003) 1422–1436.
[11] J. Cohen, Constraint logic programming languages, Communications of the ACM 33 (7) (1990) 52–68.
[12] G.E. Collins, H. Hong, Partial cylindrical algebraic decomposition for quantifier elimination, Journal of Symbolic Computation 12 (3) (1991) 299–328.
[13] G.E. Collins, J.R. Johnson, W. Krandick, Interval arithmetic in cylindrical algebraic decomposition, Journal of Symbolic Computation 34 (2) (2002) 145–157.
[14] C. Combi, M. Gozzi, J.M. Juárez, R. Marín, B. Oliboni, Querying clinical workflows by temporal similarity, in: AIME, 2007.
[15] P. Dasgupta, P.P. Chakrabarti, A. Dey, S. Ghose, W. Bibel, Solving constraint optimization problems from CLP-style specifications using heuristic search techniques, IEEE Transactions on Knowledge and Data Engineering 14 (2) (2002) 353–368.
[16] R. Davis, Diagnostic reasoning based on structure and behavior, Artificial Intelligence 24 (1984) 347–410.
[17] R. Dechter, Constraint Processing (The Morgan Kaufmann Series in Artificial Intelligence), Morgan Kaufmann, 2003.
[18] R. Dechter, I. Meiri, Experimental evaluation of preprocessing algorithms for constraint satisfaction problems, Artificial Intelligence 68 (2) (1994) 211–241.
[19] R. Dechter, J. Pearl, Network-based heuristics for constraint satisfaction problems, Artificial Intelligence 34 (1988) 1–38.
[20] J.V. den Bussche, Constraint databases, queries, and query languages, in: G. Kuper, L. Libkin, J. Paredaes (Eds.), Constraint Databases, Springer, 2000.
[21] A.D. Deo, Modeling spatial and temporal data in an object-oriented constraint database framework, Ph.D. Thesis, von der Fakultät IV-Elektrotechnik und Informatik der Technish Universität zur Erlangung des akademischem Grades, Berlin, 2002.
[22] J. Faugère, A new efficient algorithm for computing Gröbner bases without reduction to zero (f_5), ACM Press, New York, NY, USA, 2002.
[23] J.C. Faugère, A new efficient algorithm for computing Gröbner bases (f_4), Journal of Pure and Applied Algebra 139 (1–3) (1999) 61–88.
[24] R.M. Gasca, J.A. Ortega, M. Toro, A framework for semiqualitative reasoning in engineering applications, Applied Artificial Intelligence 16 (3) (2002) 173–197.
[25] P.A. Geelen, Dual viewpoint heuristics for binary constraint satisfaction problems, in: ECAI'92: Proceedings of the 10th European Conference on Artificial Intelligence, John Wiley & Sons, Inc., New York, NY, USA, 1992.
[26] F. Geerts, B. Kuijpers, On the decidability of termination of query evaluation in transitive-closure logics for polynomial constraint databases, Theoretical Computer Science 336 (1) (2005) 125–151.
[27] D. Goldin, A. Kutlu, M. Song, Extending the Constraint Database Framework, in: PCK50, ACM Press, New York, USA, 2003.
[28] D.Q. Goldin, Taking constraints out of constraint databases, in: Constraint DataBases, Lecture Notes in Computer Science, vol. 3074, Springer, 2004.
[29] L. Granvilliers, F. Goualard, F. Benhamou, Box consistency through weak box consistency, in: ICTAI'99: Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence, IEEE Computer Society, Washington, DC, USA, 1999.
[30] R. Gross-Brunschwiler, Implementation of constraint database systems using a compiler-time rewrite approach, Ph.D. Thesis, Swiss Federal Institute of Technology, Zürich, May 1996.
[31] S. Grumbach, P. Rigaux, L. Segoufin, The DEDALE system for complex spatial queries, in: SIGMOD Conference, 1998.
[32] R.M. Haralick, G.L. Elliott, Increasing tree search efficiency for constraint satisfaction problems, in: Proceedings of the Sixth International Joint Conferences on Artificial Intelligence (IJCAI), Tokyo, Japan, 1979.
[33] W. Hodges, Some strange quantifiers, in: Structures in Logic and Computer Science, 1997.
[34] X. Huang, A general extension of constraint propagation for constraint optimization, in: CP, 2004.
[35] J. Jaffar, J.-L. Lassez, Constraint logic programming, in: Principles of Programming Languages (POPL), 1987.
[36] J. Jaffar, M.J. Maher, Constraint logic programming: a survey, Journal of Logic Programming 19–20 (1994) 503–581.
[37] P.C. Kanellakis, G.M. Kuper, P.Z. Revesz, Constraint query languages, Symposium on Principles of Database Systems (1990) 299–313.
[38] J.D. Kleer, A. Mackworth, R. Reiter, Characterizing diagnoses and systems, Artificial Intelligence 56 (2) (1992) 197–222.
[39] L. Libkin, Variable independence, quantifier elimination, and constraint representations, in: ICALP'00: Proceedings of the 27th International Colloquium on Automata, Languages and Programming, Springer-Verlag, London, UK, 2000.
[40] A.K. Mackworth, Consistency in networks of relations, Artificial Intelligence 8 (1) (1977) 99–118.
[41] R. Manual, Jsolver 2.1, 2003.
[42] E. Mayol, E. Teniente, Consistency Preserving Updates in Deductive Databases, vol. 47, Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 2003.
[43] R. Reiter, A theory of diagnosis from first principles, Artificial Intelligence 32 (1) (1987) 57–96.
[44] W. Research, Mathematica 5, Reference Manual April.
[45] P. Revesz, Evaluation of queries, Introduction to Constraint Databases, Springer, 2001.
[46] P.Z. Revesz, Datalog queries of set constraint databases, in: ICDT'95: Proceedings of the Fifth International Conference on Database Theory, Springer-Verlag, London, UK, 1995.
[47] P.Z. Revesz, Safe query languages for constraint databases, ACM Transactions on Database Systems 23 (1) (1998) 58–99.
[48] P.Z. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu, Y. Wang, The MLPQ/GIS constraint database system, in: ACM SIGMOD Conference, May 16–18, Dallas, Texas, USA, ACM, 2000.
[49] P.Z. Revesz, Y. Li, Mlpq: a linear constraint database system with aggregate operators, in: IDEAS'97: Proceedings of the 1997 International Symposium on Database Engineering and Applications, IEEE Computer Society, Washington, DC, USA, 1997.
[50] M. Sachenbacher, B. Williams, Diagnosis as semiring-based constraint optimization, in: 15th International Workshop on Principles of Diagnosis, 2004.
[51] A.W. Strzebonski, Cylindrical algebraic decomposition using validated numerics, Journal of Symbolic Computation 49 (9) (2006) 1021–1038.
[52] M. Stumptner, F. Wotawa, Coupling CSP decomposition methods and diagnosis algorithms for tree-structured systems, in: International Joint Conferences on Artificial Intelligence (IJCAI), 2003.

[53] V. Telerman, Using constraint solvers in CAD/CAM systems, in: PSI'02: Revised Papers from the Fourth International Andrei Ershov Memorial Conference on Perspectives of System Informatics, Springer-Verlag, London, UK, 2001.
[54] D. Toman, SQL/TP: a temporal extension of SQL, in: Constraint Databases, Springer, 2000.
[55] P. Veltri, Constraint database query evaluation with approximation, in: ITCC'01: Proceedings of the International Conference on Information Technology: Coding and Computing, IEEE Computer Society, Washington, DC, USA, 2001.