Determination of Possible Minimal Conflict Sets Using Constraint Databases Technology and Clustering

M.T. Gómez-López, R. Ceballos, R. M. Gasca, and S. Pozo

Computer Engineering Superior Technical School of Seville, Spain
{mayte, ceballos, gasca, sergio}@lsi.us.es

Abstract. Model-based Diagnosis allows the identification of the parts which fail in a system. The models are based on the knowledge of the system to diagnose, and can be represented by constraints associated to components. Inputs and outputs of components are represented as variables of those constraints, and they can be observable and non-observable depending on the situation of sensors. In order to obtain the minimal diagnosis in a system, an important issue is to find out the possible minimal conflicts in an efficient way.

In this work, we propose a new approach to automate and to improve the determination of possible minimal conflict sets. This approach has two phases. In the first phase, we determine components clusters in the system in order to reduce drastically the number of contexts to consider. In the second phase, we construct a reduced context network with the possible minimal conflicts. In this phase we use Gröbner bases reduction. A novel logical architecture of Constraint Databases is used to store the model, the components clusters and possible minimal conflict sets. The necessary information in each phase is obtained by using a standard query language.

1 Introduction

Diagnosis allows to determine why a system correctly designed does not work as it was expected. It is based on the monitorization of a system. The diagnosis aim is to detect and to identify the reason of an unexpected behavior, or in other words, to identify the parts which fail in a system. Our proposal is based on DX [1] approaches and in other works as [2, 3]. These works were proposed to find out the discrepancies between the observed and correct behaviors of the system.

In engineering applications it is often overlooked the storage of these data and query processing. The key idea in this paper is to combine the power of Constraint Databases (CDBs) [4, 5] with the data treatment of diagnosis. We are able to improve the efficiency in some phases of the model-based diagnosis with CDBs. To improve the detection of possible minimal context conflicts, we use a program implemented in Java^{TM} and SQL (Standard Query Language). With CDBs technology [6] we are able to make the information and models persistent. Most DX approaches for components characterize the diagnosis of a system as a collection of minimal sets of failing components which explain the observed behaviors (symptoms). A conflict is a set of assumptions where, at least, one must be false. The assumptions are about behavioral modes of components. GDE [7] coupled with an ATMS [8] as inference engine uses previously discovered conflicts

C. Lemaître, C.A. Reyes, and J.A. Gonzalez (Eds.): IBERAMIA 2004, LNAI 3315, pp. 942–952, 2004.

to constrain the search in the candidate space. The most important disadvantage of using this approach is the large number of possible conflicts ($2^n - 1$), *n* being the number of components.

In this work, we propose a new approach to automate and to improve the determination of possible conflicts, it is based on:

- A structural pretreatment in order to reduce drastically the computational complexity.
- The Reduction of the number of possible contexts to treat by means of symbolic techniques in order to obtain the possible conflicts.

This technique for elimination is Gröbner bases [9] that is our projection operator to manipulate multiple polynomial variables. It eliminates the non-observable variables of the constraints of the different contexts of a previous step.

Finding all minimal conflict has been a problem active enough at last years using CS-Tree [10]. Symbolic processing algorithms (Gröbner bases) of the initial model are used by model-based diagnosis communities [11, 12]. Another proposition [13] presents the concept of a possible conflict as an alternative to the use of pre-compiled dependency-recording.

Our paper has been organized as follows: Section 2 reviews definitions and notation to allowing to formalize the subsequence operations. Section 3 shows an example to prove our solution. Section 4 describes the improvements to detect the possible minimal conflicts using CDBs and components clusters. Finally we present our conclusions and the future works in this research line.

2 Definitions and Notation

The definitions and notation used are based on the concepts proposed in the diagnosis community (DX). To introduce our work it is necessary the use of the following definitions and notation:

Definition 1. The System Polynomial Model (SPM): It can be defined as a finite set of polynomial equality constraints (*P*) which determine the system behavior. This is done by means of the relations between the non-observable variables (V_{nob}) and the observable variables (V_{ob}) which are directly obtained from sensors that are supposed to work correctly. Therefore, the tuple *SPM (*P,V_{ob},V_{nob}*)* is obtained for a system.

Definition 2. Context Set (CS): A context set of a SPM is a collection of components which compose the system. The possible context set will be $2^{comp} - 1$, where *comp* is the number of components of the system.

Definition 3. Context Network (CN): A graph formed by all the elements of the context set of the system according to the way proposed by ATMS [8].

3 System Example: A System of Heat Exchangers

In order to explain our methodology, we will apply it to the system shown in Figure 1. It was presented in [14]. This system consists of six heat exchangers, three flows f_i coming

in at different temperatures t_i. The functions of the system are described by polynomial constraints, coming from three kinds of balance:

$\sum_i f_i = 0$: mass balance at each node,
$\sum_i f_i * t_i = 0$: thermal balance at each node,
$\sum_{in} f_i * t_i - \sum_{out} f_j * t_j = 0$: enthalpic balance for each heat exchanger.

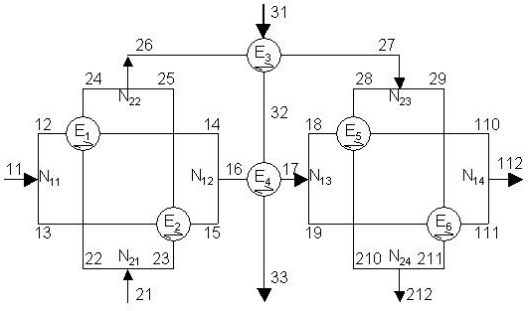

Fig. 1. System of Heat Exchangers

The system has 34 polynomial equations and 54 variables, from which 28 are observable: $t_{11}, t_{12}, t_{13}, t_{16}, t_{17}, t_{18}, t_{19}, t_{112}, t_{21}, t_{26}, t_{27}, t_{212}, t_{31}, t_{33}, f_{11}, f_{12}, f_{13}, f_{16}, f_{17}, f_{18}, f_{19}, f_{112}, f_{21}, f_{26}, f_{27}, f_{212}, f_{31}$ and f_{33}. There is no direct measure of the rest of the variables. This defines three different subsystems, each one formed by two exchangers: $\{E_1, E_2\}$, $\{E_3, E_4\}$ and $\{E_5, E_6\}$. Each of the six exchangers and each of the eight nodes of the system are considered as components whose correct functioning must be verified.

4 Computing All Possible Minimal Conflicts

The model which reflects the system structure and behavior is presented by a set of polynomial constraints. All this information is stored in a CDB.

The key idea is to generate an equivalent constraints model which has the same solution as the original one, but only with observable variables. In order to produce this model we will use Gröbner bases as symbolic technique.

4.1 Gröbner Bases

Gröbner bases theory is the origin of many symbolic algorithms used to manipulate multiple variable polynomials. It is a generalization of Gauss' elimination of multivariable linear equations and of Euclides' algorithm for one-variable polynomial equations. Gröbner bases has better computational properties than the original system. We can determine if a system can be solved or not.

The main idea is to transform the polynomial constraint set into a standard form for the resolution of problems. Having the set of equality polynomial constraints of the form $P = 0$, Gröbner bases produce an equivalent system $G = 0$ which has the same solution as the original one, but generally easier to be solved.

For our work, we have a function called GröbnerBasis, which calculates Gröbner bases by means of a finite set of polynomial equations (SPM) and a set of observable and non-observable variables.

This function allows building the context network. The signature of GröbnerBasis function looks like this:

GröbnerBasis({Polynomials}, {Observable Variables},
{Non-observable Variables})

Let us consider, for instance, the context represented by $\{N_{12}E_1E_2\}$. *Gröbner Basis* function takes the parameters:

GröbnerBasis({polynomialsOf(N_{12}, E_1, E_2)}, $\{f_{16}, f_{12}, f_{13}, t_{16}, t_{12}, t_{13}\}$,
$\{f_{14}, f_{15}, f_{22}, f_{23}, f_{24}, t_{14}, t_{15}, t_{22}, t_{23}, t_{24}\}$)

The result would be the system of polynomial constraints: $\{f_{12} + f_{13} - f_{16} = 0\}$.

4.2 Constraint Database Architecture

One of the difficulties in diagnosing a system is handling the information, therefore we have important reasons to use CDBs in model-based diagnosis:

1. By using CDBs, it is possible to add or delete some components when our system changes. In this way, rebuilding the full problem is not necessary.
2. If we do not use a CDB and the execution of the algorithm diagnosis fails, while being executed, we must reexecute the full problem because there is not partial information stored.
3. CDBs allow using the power of SQL in order to query the database and obtain the necessary information.

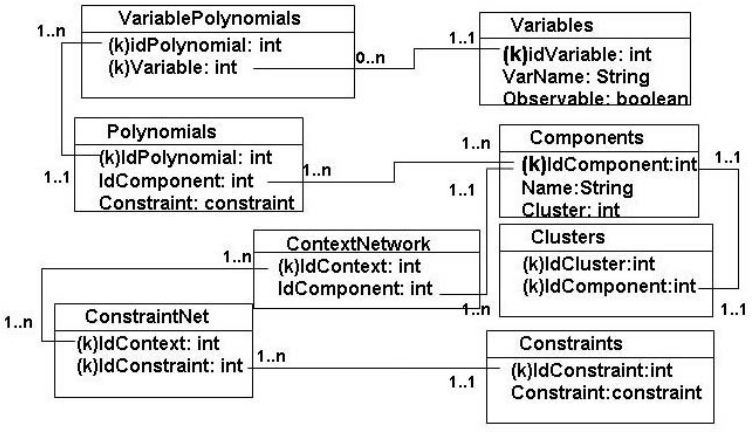

Fig. 2. Constraint Database Architecture (k: Primary Key)

First of all, we are going to explain the database architecture, and how the information is stored:

1. **Components:** This table contains the names and identifiers of the components which make up the system. In this table, the cluster identification of each component is also stored.
2. **Polynomials:** This table contains the different behaviors of the components. The components can have more than one polynomial associated.
3. **ContextNetwork:** This table represents all the relations that the process must study to obtain the minimal possible conflict context.
4. **Variables:** This table contains all the variables which participate in the system, observable and non-observable.
5. **VariablePolynomials:** This table represents the variables in each polynomial. This table is important because in order to obtain Gröbner bases we need to send the observable and non-observable variables of the polynomials.
6. **Constraints:** All the constraints are stored in this table. We will fill in this table with the GröbnerBasis function results.
7. **ConstraintNet:** This table relates each context to constraints.
8. **Clusters:** This table contains the relations between components and clusters.

4.3 First Improvement: Identification of Components Clusters

In our methodology, the first step is to isolate independent subsystems. This structural pretreatment will allow us to divide the system into independent subsystems. The possible minimal conflict sets of the system can be obtained by the conflicts of all independent subsystems. The subsystems obtained are smaller than the whole system. Therefore the computational complexity to detect conflicts from each subsystem is lower or equal than the whole system. In order to clarify the following steps we need the following definition:

Definition 4. Components cluster (CC): A set of components *C* is a components cluster, if the following predicates are true:

- All non-observable inputs and outputs of each component of *C* are always linked only to components of *C*.
- It does not exist another set *C'* with less elements than *C*, which validates the first predicate and it is included in *C*.

With the first predicate we look for the independence among conflicts of different components clusters. This predicate guarantees that it is possible to detect a conflict in a components cluster without information about other components clusters. This is possible because, in a components cluster, all the non-observable inputs and outputs are among components of the same cluster. Therefore, there is not any connection with another components cluster which is not monitored. We look for the division of our system into the biggest possible number of clusters in order to obtain a smaller computational cost. The second predicate guarantees that the components clusters will be as small as possible.

Example: For example, component E_3 is not completely monitored because we are not able to know the value of outputs f_{32} and t_{32}. Likewise, E_4 is not completely monitored because we are not able to know the value of inputs f_{32} and t_{32}. But we can monitor these two components together like they were only one component. In this case, all the inputs and outputs of this component are observable.

Algorithm: The following pseudo-code (see Figure 3) stores the set of components clusters of a system. At first, the set *C* has all the components. The algorithm extracts each time one component of *C* to create an instance (*CP*) of a components cluster. All the components from *C* which have, at least, one non-observable variable in common with one component of *CP* are added to *CP*. If it is impossible to find another component with non-observable variable in common, the components cluster is completed. The process continues with another component from *C* which has not been assigned to any components cluster. The process is finished when the set *C* is empty, and all components are assigned to one components cluster.

```
Set c = ObtainSystemComponents()
while (NotEmpty(c))
   Component x = GetComponent(c)
   Cluster cp = CreateCluster(x)
   boolean change = true
   while(change)
      Set cno = GetCommonComp(cp,c)
      deleteComponents(c,cno)
      AddComponents(cp,cno)
      Change=NotEmpty(cno)
   endwhile
   AddClusterToDB(cp)
endwhile
```

Fig. 3. Pseudocode of the components clusters algorithm

Methods to Select the Components Clusters:

- *ObtainSystemComponents():* This method returns all the system components stored in the CDB.
- *GetComponent(Set c):* It returns one component and delete it from the set *C*.
- *CreateCluster(Component x):* Here it is created a cluster with the component *x*.
- *GetCommonComp(Cluster cp, Set c):* This method returns and deletes (from the set *C*) all the components from *c* which have, at least, one non-observable variable in common with some of the components of *cp*.
- *AddComponents(Cluster cp, Set cno):* It adds all *cno* components to *cp* cluster.
- *AddClusterToDB(Cluster cp):* This method stores *cp* Cluster in the CDB.

For the example presented in Section 3, we obtain five components clusters, which are A=$\{\{N_{11}\}, \{N_{13}\}, \{N_{12},N_{21},N_{22}, E_1,E_2\}, \{N_{14},N_{23},N_{24},E_5,E_6\}, \{E_3,E_4\}\}$. For all the components clusters obtained we will build a different and independent context network. With the structural pretreatment the number of nodes is 67. Without this structural pretreatment, the number of nodes of the context network (as it appears in [12]) is 2^{14}-1.

4.4 Second Improvement: Reduction Algorithm

In order to improve the computational time in the calculation of the possible minimal set conflicts, we propose the algorithm of Figure 4. Previously we need two new definitions to understand the algorithm better.

Definition 5. *Observable Context*: It is a context with only one component and, at least, one polynomial without non-observable variables. It means that is not necessary to call GröbnerBasis function. For example $\{N_{11}\}$ and $\{E_3\}$ are observable contexts.

Definition 6. *Relevant Context*: It is a context whose components have, at least, one polynomial whose non-observable variables are also in other polynomial of the context. If we call GröbnerBasis function in other cases, we will not obtain any important results, because it is not possible to eliminate all non-observable variables from, at least, one polynomial of all the context's components:

C is a relevant context if
C$\equiv \bigcup_i \{c_i\} \mid \forall\, c_i \in \mathbf{C} \cdot \exists\, p_i \in c_i$
$\mid \forall\, \mathbf{x} \in$ **NonObsVar(**p_i**)** $\cdot\, \mathbf{x} \in \mathbf{C}$
where c_i **is a component,** p_i **a polynomial and NonObsVar(**p_i**)**
the set of non-observable variable of p_i

```
foreach (cluster in clusters)
   Set contexts=ObtainContexts(cluster)
   foreach(context in contexts)
      if(IsAnObservableContext(context))
         AddContext(context)
         UpdateTables(context)
      else
         if (RelevantContext(context))
            AddContext(context)
            CallGröbner(context)
         endif
      endif
   endforeach
endfor
```

Fig. 4. Pseudocode of the reduction algorithm for Relevant Contexts

Methods of the Reduction Algorithm:

- *ObtainContexts(Cluster cluster):* This function returns all the contexts of *cluster*
- *IsAnObservableContext(Context c):* This function returns true if the *context* is an observable context.
- *AddContext(Context c):* This function adds the context *c* to table ContextNetwork .
- *UpdateTables(Context c):* This function stores all the polynomial constraints of the context *c*, in the tables ConstraintNet and Constraint. Here it is not necessary to call GröbnerBasis because the polynomials do not have any non-observable variables.

Table 1. CARCs

Index	CC	Constraints
1	1	f11 - f12 - f13
2	1	-f11 t11 + f12 t12 + f11 t13 - f12 t13
3	2	f17 - f18 - f19
4	2	-f17 t17 + f18 t18 + f17 t19 - f18 t19
5	3	f12 + f13 - f16
6	3	f21 - f26
7	3	f12 t12 + f13 t13 - f12 t16 - f13 t16 + f21 t21 - f21 t26
8	4	f18 + f19 - f112
9	4	f27 - f212
10	4	f18 t18 + f19 t19 - f18 t112 - f19 t112 + f27 t27 - f27 t212
11	5	f26 - f27
12	5	f16 - f17
13	5	f31 - f33
14	5	f16 t16 - f17 t17 + f26 t26 - f27 t27 + f31 t31 - f31 t33

- *RelevantContext(Context c):* This function returns true if the context *c* is a relevant context.

  Example: *Context:* N_{22}**, E_1 *and* E_2**
  In this case the component E_1 do not have any polynomial with all non-observable variable couple with other component

To implement this idea, we propose a query to know what are the non-observable variables of a *polynomial* **p** which are also in the same *context* **c** but in different polynomial.

```
SELECT DISTINCT v.VARNAME
FROM VARIABLES v, VARIABLES v2,
     VARIABLEPOLYNOMIALS cv, POLYNOMIALS c,
     VARIABLEPOLYNOMIALS cv2, POLYNOMIALS c2,
     CONTEXTNETWORK rc, CONTEXTNETWORK rc2,
WHERE c.ID=p AND
     c.IDCOMPONENT=rc.IDCOMPONENT AND
     rc.ID=c AND c.ID=cv.ID
     AND cv.VARIABLE=v.IDVARIABLE AND
     v.OBSERVABLE=false AND c.ID<>c2.ID AND
     rc2.ID=rc.ID AND c2.ID=cv2.ID AND
     c2.IDCOMPONENT=rc2.IDCOMPONENT AND
     cv.VARIABLE=cv2.VARIABLE
```

Comparing the query result and the non-observable variable of the polynomial, we will know if all the non-observable variables of a polynomial are in another polynomial, and therefore if it is a relevant context.

– *CallGröbner():* We build GröbnerBasis function call with information from the tables ContextNetwork, Polynomials, VariablePolinomial and Components. The results will be stored, if they are not in the table Constraints. Finally, the constraint and the corresponding context will be stored in the table CostraintNet.

Table 2. Improvement using components cluster and relevant context

	No reduction	Using CC	Using CC and RC
Number of Contexts	2^{14}-1	67	67
Calls to GB. function	2^{14}-1	67	7
Obtained Constraints	64	14	14
Elapsed time	4'2 days	7 Seconds	1 Second

(This test have been carried out in a Pentium IV-2Ghz with 512 MB)

With our solution, we only create 11 contexts and we only call GröbnerBasis function 7 times, because the contexts $\{N_{11}\}$, $\{N_{13}\}$, $\{E_3\}$ and $\{E_4\}$ are observable contexts. The reduced algorithm obtains 14 constraints which are shown on Table 1. Table 2 shows the differences among using all contexts (as in [12]) and using our approach.

4.5 Determination of Possible Minimal Conflict Contexts

In order to determinate the possible minimal conflicts we apply a constraint-driven algorithm. The following definition is necessary in this process:

Definition 7. Context Analytical Redundancy Constraint (CARC): It is a constraint derived from SPM, in such a way that only the observable variables are related.

In our approach, the set of CARCs of the system (Table 1) is the union of all the constraints. These constraints were obtained in each components cluster in two ways, directly from observable context or using GröbnerBasis function to each relevant context.

All the relevant context are shown in Figure 5 for the Heat Exchangers example.

5 Conclusions and Future Works

This paper proposes a new approach to automate and to improve the determination of possible minimal conflict sets. The determination of components clusters of the system reduces the number of contexts to consider. Only the relevant contexts are studied in order to reduce the computational complexity. In this paper we propose a CDB architecture to store polynomial constraints using standard SQL and Java^{TM} language to obtain and handle the constraint information. Another advantage is the power of SQL storing and getting information in CDBs.

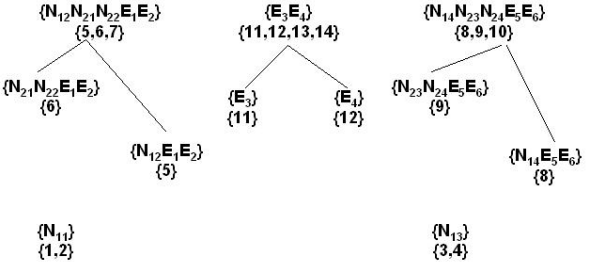

Fig. 5. Possible Minimal Conflict Network of the System

Extension to ODEs with polynomial constraints, in order to deal with dynamic systems, is our next objective. At the same time it is interesting for future works to study how the minimal context network changes when some polynomials change, and to look for techniques to avoid restudying all the system. In other way, there is a wide field to study the diagnosis of systems with components are located in different CDBs.

Acknowledgements

This work has been funded by the M. de Ciencia y Tecnología of Spanish (DPI2003-07146-C02-01) and the European Regional Development Fund (ERDF/ FEDER).

References

1. Davis, R.: Diagnostic reasoning based on structure and behavior. In Artificial Intelligence 24 (1984) 347–410
2. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32 **1** (1987) 57–96
3. Kleer, J.D., Mackworth, A., Reiter, R.: Characterizing diagnoses and systems. Artificial Intelligence 56 **2-3** (1992) 197–222
4. Goldin, D., Kanellakis, P.: Constraint query algebras constraints. Journal E. F. editor (1996)
5. P. C. Kanellakis, G.M.K., Revesz, P.Z.: Constraint query languages. Symposium on Principles of Database Systems (1990) 299–313
6. Revesz, P.: Introduction to Constraint Databases. Springer (2001)
7. Kleer, J.D., Williams, B.: Diagnosing multiple faults. Art. Int. (1987)
8. Kleer, J.D.: An assumption-based truth maintenance system. Artificial Intelligence 28 **2** (1986) 127–161
9. Buchberger, B.: Gröbner bases: An algorithmic method in polynomial ideal theory. Multidimensional Systems Theory, N. K. Bose, ed. (1985) 184–232
10. de la Banda, M.G., Stuckey, P., Wazny, J.: Finding all minimal unsatisfiable subsets. Proc. Of the 5th ACM Sigplan Internacional (2003)
11. Frisk, E.: Residual generator design for non-linear, polynomial systems - a gröbner basis approach. In Proc. IFAC Safeprocess, Budapest (2000)

12. Gasca, R., Valle, C.D., Ceballos, R., Toro, M.: An integration of fdi and dx approaches to polinomial models. 14th International Workshop on principles of Diagnosis - DX (2003)
13. Pulido, J.: Posibles conflictos como alternativa al registro de dependencias en línea para el diagnóstico de sistemas continuos. PhD. degree, Universidad de Valladolid (2000)
14. Guernez, C., Petitot, M., Cassar, J., Staroswiecki, M.: Fault detection and isolation on non linear polynomial systems. 15th IMACS World Congress (1997)