DxPCs: An integrated package for model-based diagnosis of dynamic systems using Possible Conflicts

Belarmino Pulido[1], Carlos Alonso-González[1], Anibal Bregon[1],
Alberto Hernández[1], and Luis Miguel Villarroel[1]

Departamento de Informática. Universidad de Valladolid. Valladolid. Spain
{belar,calonso,anibal}@infor.uva.es

Abstract. This work introduces DXPCS, a software tool performing model-based diagnosis of continuous dynamic systems whose models can be represented as a set of Algebraic/Ordinary Differential Equations. The diagnosis approach implemented is based upon the Possible Conflict (PC) concept. DXPCS is mainly intended for educational purposes, providing a complete package to show the Artificial Intelligence approach to model-based diagnosis for postgraduate students. Given a set of equations, together with structural information about the model, DXPCS is able to automatically build simulation models for each PC, it can handle both single-fault and multiple-fault scenarios, for both parametric and additive faults. Different options for fault detection, residual generation and evaluation can be chosen. The software architecture and the software performance for one simple case study, are provided in this paper.

Keywords: Model-based diagnosis, dynamic systems diagnosis, consistency-based diagnosis

1 Introduction

Model-based diagnosis fundamentals rely upon the idea of comparing the estimated behaviour of a system, given a model, with real observed behaviour. Since the early days of model-based diagnosis there have been software tools implementing different diagnosis proposals [8]. Such trend is still active and nowadays several tools are available to test recent proposals in the field, from both the Artificial Intelligence –Raz'r [16], TRANSCEND [11], Squal-track [1], LyDIA-NG [6], Hyde [12]–, and the Control Theory communities – Symbols [2], FlexDx [9], or other Matlab toolboxes [7]–. Moreover, several tools have focused only on the structural aspects of the model-based diagnosis process, related to find the set of over-determined submodels[1] required to perform the fault detection and isolation stages.Later on, those submodels can be used to estimate system behaviour as simulation or state-observer models, for instance.

This work introduces DXPCS, Diagnosis with Possible Conflicts, which implements a comprehensive equation-based approach to model-based diagnosis, including fault detection and isolation capabilities for continuous systems diagnosis. The models must be made up of a set of Algebraic and/or Ordinary Differential Equations, ADEs

[1] An over-determined set of equations provides analytical redundancy, which is the essence of model-based diagnosis.

or ODEs for short, the sets of input and output measurements, together with a description of the set of state and intermediate variables, and the set of parameters related to faults. DXPCS is able to generate an input/output model given the set of ADEs/ODEs. Such model can be later used to generate the set of Possible Conflicts, or PCs for short, that are the basis for the diagnosis approach. The tool is capable to generate new executable submodels, one for each PC. An additional feature is that DXPCS can generate a set of diagnosis scenarios with varying input conditions, for both nominal and faulty situations. These scenarios can be stored or studied in a consistency-based diagnosis framework using PCs. As a consequence, DXPCS provides a complete package intended mainly for educational purposes for postgraduate students.

The rest of the paper is organized as follows. First we introduce the main concepts underlying the diagnosis approach implemented in DXPCS, and a case study. Later on we summarize the main functionalities of DXPCS. Finally, we discuss DXPCS features against related work.

2 The diagnosis approach

In this section we summarize consistency-based diagnosis using PCs, together with a running example that will be used to illustrate how DXPCS works.

In this work we will only talk about dynamic continuous systems, with both nominal and faulty states, and whose behaviour is described by the following kind of model.

Definition 1 (Model). *The system model is defined as $M(\Sigma, U, Y, X, Z, \Theta)$, where: Σ is a set of ADEs/ODEs, defined over a collection of known and unknown variables: U is the set of inputs, Y is the set of outputs, X is the set of state variables, and Z is the set of intermediate variables. Finally, Θ is the set of parameters*[2].

An implicit assumption in our modelling approach is that we can use the same set of equations for both the nominal and the faulty behaviour estimation, just changing the value of some model parameters: $\theta_i \in \Theta$.

In this work, we will use the three-tank system shown in Fig. 1 as a running example.

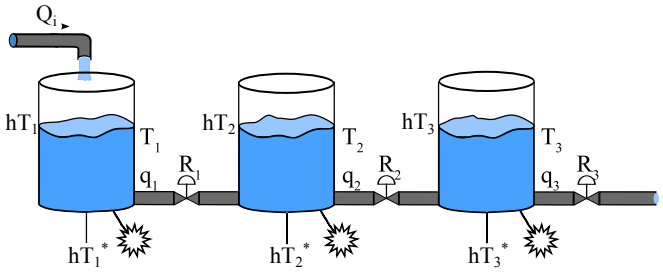

Fig. 1: Diagram of the three-tank system.

[2] Since we are dealing with fault diagnosis, in our model we are mainly interested in every parameter suitable to model faulty behaviour.

The three identical tanks are denoted as T_1, T_2, and T_3. Q_i is a measured input flow for tank T_1, which is drained into T_2 via pipe with flow q_1. The liquid level is denoted as hT_1. A similar process gets the flow from $T2$ to $T3$ via pipe with flow q_2. Finally, there is an output flow q_3 from T_3. Tanks height are measured by sensors $\{hT_1^*, hT_2^*, hT_3^*\}$.

The following set of ADEs models the behaviour of the system. The change in the height in each tank, d_hT_i, is computed according to mass balances:

$$d\_hT_1/dt = (Q_i - q_1 - qf_1)/A_1 \tag{1}$$
$$d\_hT_2/dt = (q_1 - q_2 - qf_2)/A_2 \tag{2}$$
$$d\_hT_3/dt = (q_2 - q_3 - qf_3)/A_3 \tag{3}$$

Flows between tanks, q_i, and leakage flows, qf_j are modelled using the following equations, where parameters $Stuck_{T_i}$ and $Leakage_{T_i}$ are used to model blockages in pipes and tank leakages, respectively. In this way, we handle both additive and multiplicative faults in the model.

$$q_1 = Stuck_{T1} \times \sqrt{\mid hT_1 - hT_2 \mid}/Rv_1 \tag{4}$$
$$qf1 = Leakage_{T1} \times \sqrt{hT_1} \tag{5}$$
$$q_2 = Stuck_{T2} \times \sqrt{\mid hT_2 - hT_3 \mid}/Rv_2 \tag{6}$$
$$qf2 = Leakage_{T2} \times \sqrt{hT_2} \tag{7}$$
$$q_3 = Stuck_{T3} \times \sqrt{\mid hT_3 \mid}/Rv_3 \tag{8}$$
$$qf_3 = Leakage_{T3} \times \sqrt{hT_3} \tag{9}$$

It is assumed that the initial water level in the three tanks is zero, but this can be changed using an initial condition file for the simulation. Additionally, we make explicit the relation between the state variables, hT_i in our example, and their derivatives d_hT_i, through equations (10), (11), and (12), for tanks T_i with $i = 1, 2, 3$. These three equations allow us to select an integral or differential approach for behaviour simulation, depending on the selected causality. These equations will not introduce faulty behaviour in the system, because they have no $\theta_i \in \Theta$.

$$hT_i = \int d\_hT_i \cdot dt \qquad (10), (11), (12)$$

Finally, the observational model Y, relates unknown and measured variables through equations (13), (14), and (15), for tanks T_i with $i = 1, 2, 3$.

$$hT_i^* = hT_i \qquad (13), (14), (15)$$

2.1 Consistency-based diagnosis with PCs

As previously mentioned, model-based diagnosis uses one model of the system to estimate expected correct behaviour. This behaviour is compared against available observations looking for a discrepancy or residual signal, which is used for fault detection and isolation purposes, and it is formally defined as follows:

Definition 2 (Residual). *A residual is a real-valued measure $R(y_i, \hat{y_i})$ of the difference between real and estimated system output at time t.*

The goal of the diagnosis system is to determine the health status of every parameter in the model: $\theta_i \in \Theta$ to be ok or $faulty$.

Definition 3 (Diagnosis). *A diagnosis for the system is made up of* (M, Π)*, where*

- $M(\Sigma, U, Y, X, Z, \Theta)$ *is a system model, and:*
- Π *is a mapping function:* $\Theta \rightarrow \{ok, faulty\}^n$.

Consistency-based Diagnosis, or CBD, only requires correct behaviour models to perform fault detection and isolation[3]. The process is organized around the conflict concept [8], corresponding to a discrepancy detection. Faulty candidates can be computed as the minimal-hitting sets of those components, or correctness assumptions linked to them, involved in conflict detection. This work relies upon Possible Conflicts [14], or PCs, to avoid on-line computation of the set of correctness assumptions in conflicts. PCs are designed to find off-line those subsystems capable of becoming conflicts online.

To compute the set of PCs we require an abstraction of $M(\Sigma, U, Y, X, Z, \Theta)$: the structural model that only relates constraints and variables in the equation models. According to [14] the structural model $H_{SD} = \{V, R\}$, where V is the set of variables, and R is the set of relations among those variables:

- $V = OBS \ \cup \ NOBS$; $\ OBS = U \cup Y$, is the set of observed or measured variables, and $NOBS = X \cup Z$, is the set of internal variables;
- $R = \{r_1, \ldots, r_k\}$, where each r_i abstracts a relation among variables $v_i \subseteq V$, as described by each equation $\sigma_i \in \Sigma$.

PCs computation proceeds in two steps:

1. Searching for any minimally overdetermined subsets of equations in H_{SD}. Each subset is called a *Minimal Evaluable Chain*, or MEC, and does not include any information about potential causal assignments for each relation r_k in MECs.
2. For each MEC, every potential causal assignment for each r_k is considered. If there is, at least, one globally valid causal assignment, it is called a *Minimal Evaluable Model*, or MEM.

Each MEM represents a set of overdetermined sets of equations with minimal analytical redundancy. Hence, it can be used to perform fault detection. Each MEM can be implemented either as a simulation or state-observer model and the global causal assignment allows to track exactly one observed variable in the original system model.

Definition 4 (Possible Conflict). *The set of constraints in a MEC that gives rise to at least one MEM is a Possible Conflict.*

We are only interested in those equations containing parameters that can be used to model faulty behaviour. Hence, we associate to each PC_k the set of parameters, $\Theta_k \subseteq \Theta$, that appears in the original equations described by each MEM_k. If we consider that each $\theta_{k_i} \in \Theta_k$ can have either ok or $faulty$ values, they can be considered as the set

[3] Parameters for modelling stuck pipes or leakages are included in DXPCS to simulate diagnosis scenarios.

of correctness assumptions for each PC. Once a MEM tracking the behaviour of one subsystem detects a discrepancy, its corresponding PC is activated as a real conflict. Afterwards, we can obtain the set of fault candidates as the minimal hitting-set of their sets of correctness assumptions.

The set of PCs for the three tank system is shown in Table 1.

Table 1: Set of PCs for the three tank system. Parameters S_{T_i} and L_{T_i} applies for $Stuck_{T_i}$, and $Leakage_{T_i}$ respectively.

PC	Equations	Parameters	Output
PC_1	$\{1, 4, 5, 10, 13, 14\}$	$\{S_{T_1}, L_{T_1}\}$	${hT_1}^*$
PC_2	$\{2, 4, 6, 7, 11, 13, 14, 15\}$	$\{S_{T_1}, L_{T_1}, S_{T_2}, L_{T_2}\}$	${hT_2}^*$
PC_3	$\{3, 6, 8, 9, 12, 14, 15\}$	$\{S_{T_2}, L_{T_2}, S_{T_3}, L_{T_3}\}$	${hT_3}^*$

3 DXPCS architecture

DXPCS is a model-based reasoning tool that implements consistency-based diagnosis, CBD, of continuous dynamic systems. The basis for model-based reasoning is a system model, M, as defined above, which is given as input.

DXPCS integrates several packages that previously worked independently to perform fault detection and isolation, or that previously required human manipulation of the models[13, 15, 4, 3, 5]. DXPCS organizes the diagnosis process around the structural description for each PC, derived from M. This description must be provided as an additional input (this file can be automatically generated using the available Java tool for computing PCs [10][4]).

3.1 Functional architecture

DXPCS conceptual model is summarized in Figure 2:

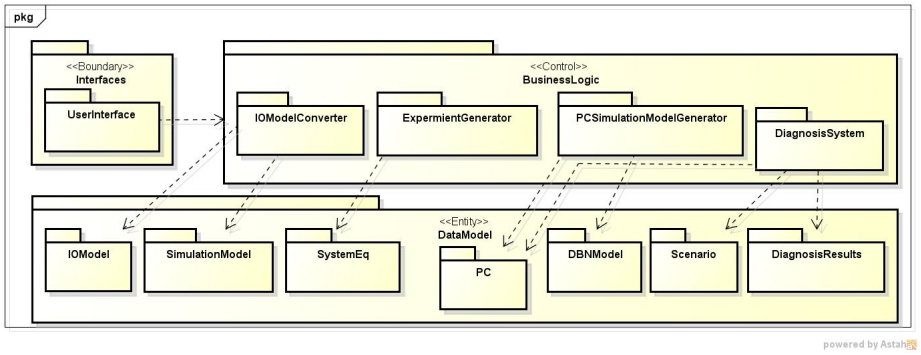

Fig. 2: Conceptual model of DXPCS software architecture.

[4] http://www.infor.uva.es/~belar/SoftwareCPCs/PCs3.0_Setup.exe

– It is able to generate Matlab simulation models, for both the complete set of ODEs, and for the PCs structural models.
– It can simulate different nominal and fault scenarios –including single and multiple parametric or additive faults–, with noisy time-varying input signals.
These results can be visualized or stored in a file or a Data Base, for later use in the diagnosis module.
– The diagnosis module implements a PC-based approach for continuous systems, including fault detection and localization capabilities. Diagnosis results can be visualized or stored in a different Data Base.

3.2 From state-space equations to PCs simulation models

Once the user chooses the *Load model* command, she/he must provide the required system description or *System Model* made up of the following files: the set of ADEs/ODEs, the set of state variables, X, the set of fault parameters, Θ, and the set of input, U, and output, Y, measurements. Afterwards, the DXPCS interface will look like the left-hand side in Figure 3, where the system model for the three tank case study has been loaded. As it can be seen, the model is made up of a set of equations involving algebraic expressions between internal, state and input variables. Matlab functions, such as $sqrt()$ or $abs()$, can also be used. These equations correspond to the instantaneous constraints in the model, providing a valid causal assignment that can be used to simulate the complete system behaviour. They correspond to equations (1) to (9) in Section 2.

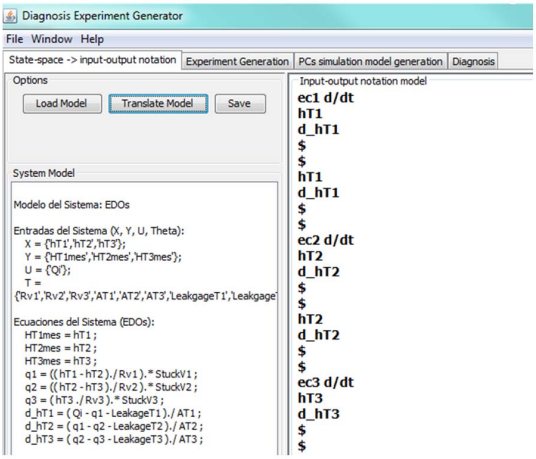

Fig. 3: Given a behavioural model as a set of ADEs, DXPCS is able to automatically generate the structural model, required to compute the set of Possible Conflicts.

There is no explicit presence of the time index. Later on, depending on the simulation method, they will be completed with one additional equation for each state

variable: in the model in Figure 3 there are three state-variables: $\{hT1, hT2, hT3\}$, therefore there must be an explicit equation to model the transient behaviour for each state-variable. For instance, the instantaneous change in the state-variable $hT1$ is modelled by equation (1), which explicitly includes its change over time, d_hT1:

$$d\_hT1 = (Qi - q1 - qf1)/AT1$$

Finally, the value of output variables –i.e. the observational model– must be explicitly included. For instance, if $hT1$ is measured, its corresponding output equation is $HT1mes = hT1$[5]. These are equations (13) to (15) in Section 2.

As can be seen in the left hand side of Figure 3, equations (10) to (12) are not included, since they depend on the particular inference engine, and they will be added automatically by the tool.

It should be noticed that DXPCS right now does not perform any syntactic or numerical analysis in the model. We assume that the model is correct and it is parameterized accordingly for the diagnosis stage.

If we choose the *Convert* option, the ODE description of the system will be converted to the input-output notation that can be seen in the right-hand side of Figure 3. This is the structural model, hence it only contains the name of the equation and the name of the state and internal variables involved in the equation. For each equation, there must be at least a causal interpretation. The default causal interpretation is given in the ADE/ODE file. If we want to include other optional causal assignments, we can edit the structural model and include them by hand. Such structural model is the input for the tool that compute the set of PCs, CPCVall [10].

3.3 Generation of PCs simulation models

This option creates a simulation model for each PC, given the system description and the PCs structural models.

Figure 4 shows on the left-hand side both the complete *System Model*, and the *Structural Model* for each PC found in the system.

On the right-hand side of the figure we can observe the set of equations required to simulate PC_1, PC_2, and PC_3, in the case study example. The subset of equations have been ordered according to the structural model. The last equation is the computation of the only output variable in the system: hT_1^* for PC_1, hT_2^* for PC_2, and hT_3^* for PC_3. The comparison of these values against those provided by the simulation will define the set of discrepancies or residuals in our diagnosis system. Each PC_i will be sensitive only to those $\theta_{pc_i} \in \Theta$. For instance, PC_1 is sensitive to faults $\{StuckV1, LeakageT1\}$, representing a blockage in valve $V1$, and a leakage at the bottom of tank T_1, respectively.

3.4 Studying different diagnosis scenarios

Before we proceed to the diagnosis stage, it is necessary to generate different diagnosis scenarios. DXPCS is able to simulate the *System Model* in different working conditions:

[5] We use the $HT1mes$ notation in Matlab instead of the hT_1^* notation to avoid confusion between asterisk and multiplication operator.

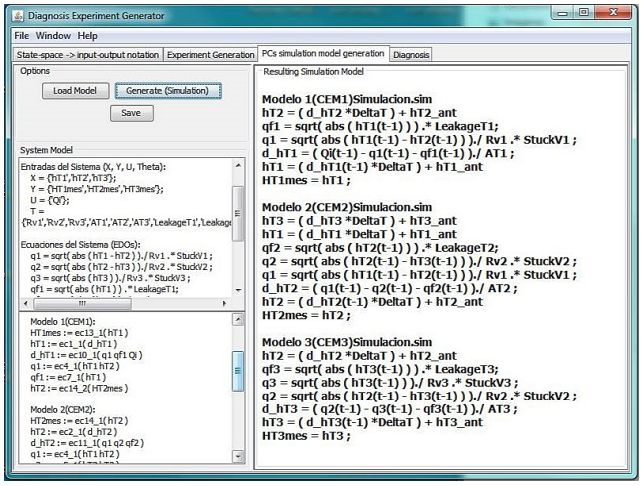

Fig. 4: Given the structural model, and the set of Possible Conflicts, DXPCS is able to automatically generate the simulation model for each Possible Conflict.

- We can generate or use an input file with different kind of signals for every variable $u_i \in U$: with or without noise, fixed or periodically changing values. We must also set the simulation period, and the mean value for each input variable, $u_i \in U$.
- Fault profiles for each scenario are very simple. Single and multiple fault scenarios are allowed. We must provide the time instant, the ordinal number i for faulty parameter $\theta_i \in \Theta$, and the fault magnitude.
- Finally, the initial conditions for simulation must be provided: the value of the state variables for the initial integration step, and the nominal value for each $\theta_i \in \Theta$

Once the scenario is defined, DXPCS relies upon Matlab to perform the simulation of the whole system model for the entire simulation period. The results are stored in a text file or in an application specific Data Base.

To perform a *Diagnosis* experiment, we must first load a system model, then load the set of PC models that we want to use in the experiment, and then load a diagnosis scenario. It should be noticed that real data can be used as an input. The only current requirement is to be stored in the scenario data base with the proper format. In this current version DXPCS does not include any automatic filtering for the input data. This could be included as part of the system model if needed.

Once the input data are available, both the residual generation algorithm and the fault detection policy can be customized. For instance, we can select a basic Euclidean distance for residual generation, together with an statistical test for residual activation (in our case we use a customized z-test).

Once those parameters are set, the system run the PC simulation models using as inputs the diagnosis scenario outputs. In every simulation step a residual is computed, and the residual analysis is performed. Whenever a residual activation is triggered, the set of faulty components are computed using an incremental version of the minimal-hitting set algorithm. Results for a T1 leakage are shown in Figure 5.

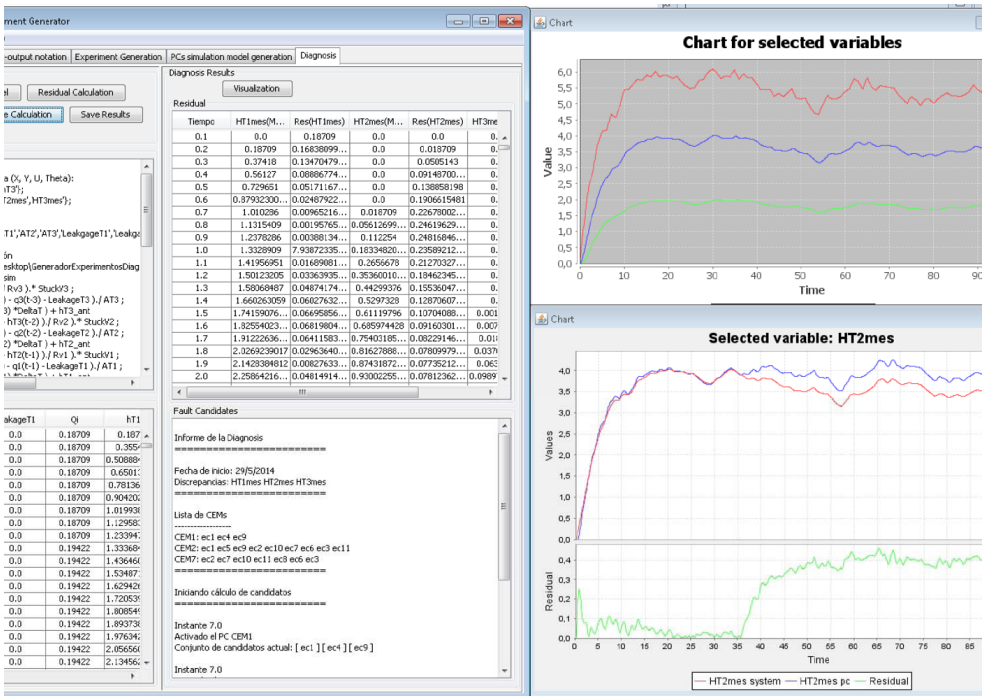

Fig. 5: DXPCS interface for a diagnosis experiment: evolution of measured and estimated variables, residual computation and fault diagnosis candidates.

4 Discussion

DXPCS is a Java tool able to simulate different diagnosis scenarios, and to perform consistency-based diagnosis of continuous systems using Possible Conflicts. DXPCS is equation oriented, and currently requires the presence of the Matlab environment in order to perform numerical simulation.

DXPCS is rather similar to traditional FDI approaches [7] –except for the kind of structural decomposition method and the residual generation–, but different from other tools relying upon other modelling approaches such as Bond-Graph [2, 11] or component-oriented modelling approaches [16].

Currently DXPCS is able to perform diagnosis using pure numerical simulation, but it is straightforward to generate state-observers given the set of PCs, and to implement them as particle-filters. As a consequence, it is different from other tools relying upon some kind of qualitative simulation or estimation (such as Raz'r [16], or TRANSCEND [11]), or interval-based simulations, such as Squal-track [1].

The current software framework does not allow hybrid behaviour modelling (as in TRANSCEND [11], LyDIA-NG [6], or Hyde [12]). As further work we plan to extend the models to a certain type of hybrid systems models, compatible with the hybrid Possible Conflicts approach to hybrid systems diagnosis [3], and also will be extended to introduce fault prognosis capabilities. Additional capabilities regarding on-line diagnosis coupling DXPCS with a real-time data base will be included too.

Acknowledgments

This work has been supported by Spanish MINECO DPI2013-45414-R grant.

References

1. Armengol, J., Vehi, J., Sainz, M., Herrero, P., Gelso, E.: Squaltrack: A tool for robust fault detection. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 39(2), 475–488 (April 2009)
2. Bouamama, B.O., Samantaray, A., Medjaher, K., Staroswiecki, M., Dauphin-Tanguy, G.: Model builder using functional and bond graph tools for fdi design. Control Engineering Practice 13(7), 875 – 891 (2005)
3. Bregon, A., Alonso, C., Biswas, G., Pulido, B., Moya, N.: Fault diagnosis in hybrid systems using possible conficts. In: Proc. of Safeprocess'12. Mexico City, Mexico (2012)
4. Bregon, A., García, D.: A reconfigurable system for multiple off-line simulation supporting fault diagnosis tasks. Information systems. Problems, perspectives, innovation approaches. pp. 39–43 (2007)
5. Bregon, A., Biswas, G., Pulido, B.: A decomposition method for nonlinear parameter estimation in transcend. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on 42(3), 751–763 (2012)
6. Feldman, A., de Castro, H.V., van Gemund, A., Provan, G.: Model-based diagnostic decision-support system for satellites. In: Aerospace Conference, 2013 IEEE. pp. 1–14. IEEE (2013)
7. Frisk, E., Krysander, M., Nyberg, M., Aslund, J.: A toolbox for design of diagnosis systems. In: IN PROC. OF IFAC SAFEPROCESS'06 (2006)
8. Hamscher, W., Console, L., de Kleer, J.: Readings in Model-based Diagnosis. Morgan-Kaufmann Pub., San Mateo (1992)
9. Heintz, F., Krysander, M., Roll, J., Frisk, E.: FlexDx: A Reconfigurable Diagnosis Framework. In: Proceedings of the 19th International Workshop on Principles of Diagnosis, DX08. pp. 79–86. Blue Mountains, Australia (September 2008)
10. Lajo, R.: Herramienta Configurable para el Cálculo del Conjunto de Posibles Conflictos. Dpto. de Informática. E.T.S.I. Informática. Universidad de Valladolid (2007)
11. Manders, E., Biswas, G., Mahadevan, N., Karsai, G.: Component-oriented modeling of hybrid dynamic systems using the Generic Modeling Environment. In: Proceedings of the 4th Workshop on Model-Based Development of Computer Based Systems (2006)
12. Narasimhan, S., Brownston, L.: Hyde–a general framework for stochastic and hybrid model-based diagnosis. In: Proc. 18th International Workshop on Principles of Diagnosis (DX'07), Nashville, USA. pp. 162–169. Citeseer (2007)
13. Pulido, B., Alonso, C., Acebes, F.: Lessons learned from diagnosing dynamic systems using possible conflicts and quantitative models. In: Engineering of Intelligent Systems. XIV Conf. IEA/AIE-2001. LNAI, vol. 2070, pp. 135–144. Budapest, Hungary (2001)
14. Pulido, B., Alonso-González, C.: Possible Conflicts: a compilation technique for consistency-based diagnosis. IEEE Trans. on Systems, Man, and Cybernetics. Part B: Cybernetics 34(5), 2192–2206 (Oct 2004)
15. Pulido, B., Bregon, A., Alonso-Gonzalez, C.: Analyzing the influence of differential constraints in Possible Conflict and ARR computation. In: Current Topics in Artficial Intelligence, CAEPIA 2009 Selected Papers. P. Meseguer, L. Mandow, R. M. Gasca Eds. Springer-Verlag Berlin (2009)
16. Struss, P., Price, C.: Model-based systems in the automotive industry. AI Mag. 24(4), 17–34 (2004)