MEDITO: A logic-based meta-diagnosis tool

Nuno Belard [*†‡], Yannick Pencolé [†‡] and Michel Combacau [†‡]
[*] *Airbus France; 316 route de Bayonne; 31060 Toulouse, France*
[†] *LAAS-CNRS; 7 Avenue du Colonel Roche; F-31077 Toulouse, France*
[‡] *Université de Toulouse; UPS, INSA, INP, ISAE; LAAS; 7 Avenue du Colonel Roche, F-31077 Toulouse, France*
nuno.belard@airbus.com, ypencole@laas.fr, combacau@laas.fr

***Abstract*—In every Model-Based Diagnosis (MBD) approach, a model of a real-world system and some observations of such a system are used by a diagnostic algorithm to compute diagnoses. Contrary to MBD classical hypotheses, real-world applications provide us with empirical data suggesting that diagnostic systems, i.e. a model, observations and a diagnostic algorithm, are sometimes abnormal with respect to some required properties. This is where Meta-Diagnosis comes into play with a theory to determine abnormalities in diagnostic systems. Unfortunately, Artificial Intelligence lacks of a tool putting meta-diagnosis theory to practice. Our first contribution in this paper is such a tool, called MEDITO. Moreover, we provide a real-world example of MEDITO's application at meta-diagnosing an Airbus landing gear extraction and retraction system with successful results.**

***Keywords*-Model-Based Diagnosis; Meta-Diagnosis; Tool; Airbus Landing Gear Extraction and Retraction System;**

I. Introduction

In model-based diagnosis (MBD) [1][2], a diagnostic algorithm uses a model of a real-world system under study (hereafter referred to as *believed system* [1]) along with some observations gathered from it to compute diagnoses, i.e. to determine the normal and abnormal physical units of the real-world system.

Unfortunately, and contrary to classical assumptions, the tuple (believed system, observations, diagnostic algorithm), called *diagnostic system*, can be abnormal with respect to some properties one wants it to have, such as, for example: the truth and/or completion of believed systems, the soundness and/or completeness of diagnostic algorithms or the truth of observations; with a possible direct impact on the quality of the computed diagnoses. Finding such abnormalities constitutes a problem of meta-diagnosis, for which a theory is provided in [3].

Meta-diagnostic problems are ubiquitous in real-world applications. At Airbus, for example, with so many different, highly-customizable and evolving aircraft, a lot of effort is required for engineers to produce and maintain a true believed system - a necessary (although insufficient) condition to ensure efficient fault isolation procedures with no delays - by manually finding abnormalities in believed systems. More examples come from SymCure users [4], who struggle to obtain true believed systems and true observations; for these are constraints imposed by SymCure's diagnostic algorithm.

If, on the one hand, [3] provides us with a theory of meta-diagnosis enabling the formulation and theoretical solution of meta-diagnostic problems; on the other hand, Artificial Intelligence still lacks of a meta-diagnostic tool where real-life meta-diagnostic problems such as the ones mentioned above can be solved. We answer such need by contributing, in Section III, with a detailed architectural and functional description of a logic-based MEta-DIagnostic TOol (MEDITO), based on the referred meta-diagnostic framework.

Having MEDITO at our disposal, we also contribute, in Section IV, by meta-diagnosing an Airbus Landing Gear Extraction and Retraction System (LGERS) diagnostic system, and providing some positive results obtained.

Finally, since such contributions rely on the frameworks of diagnosis and meta-diagnosis of [1], [2] and [3], Section II will be dedicated to an overview of such frameworks.

II. Preliminaries

Before going any further, we assume that the reader is familiar with model-theoretic notions of *structure*, *model* and *extensions* [5]; as well as with the model-based diagnosis framework described in [1] and [2].

Since sections III and IV rely on the theories of diagnosis and meta-diagnosis, as well as in diagnostic system and diagnostic results' properties and their relations, we provide, along the present section:

- an overview of the theory of diagnosis introduced by Reiter in [1] and de Kleer and Williams in [2],
- a brief description of the theory of meta-diagnosis proposed by Belard, Pencolé and Combacau in [3], and
- some of the diagnostic system and diagnostic results' properties and property relations announced by the latter authors in [6] and [3].

A. Characterising diagnoses

Model-based diagnosis is a reasoning problem that aims at retrieving system abnormalities given a system description (the so-called SD) and a set of observations OBS. For such a couple (SD,OBS), the crucial assumption of MBD is that (SD,OBS) *matches* with the underlying reality (i.e. the real system and the real observations). More formally

[1]The word "model" will be reserved for a model-theoretic context.

speaking, reality is only accessible through a structure, let us say Ψ, of raw information everyone would have access to (in terms of behaviour and observations) if engineering and computational resources were unlimited; and the principle of model-based diagnosis is that there always exists a structure s, model of $\mathsf{SD}\cup\mathsf{OBS}$ (i.e. $\mathsf{s}\in\mathsf{Mod}(\mathsf{SD}\cup\mathsf{OBS})$), that can be extended to a structure t, i.e. $\mathsf{s}\subseteq\mathsf{t}$, which is isomorphic to Ψ, denoted $t \rightleftharpoons \Psi$. In Tarki's terms [7], MBD relies on the fact that $\mathsf{SD}\cup\mathsf{OBS}$ is an *ontologically true* theory "which says that the state of affairs is so and so, and the state of affairs is indeed so and so". In this paper, we call *real system* a set R of interacting Replaceable Units (RU), where maintenance actions resulting from diagnosis take place. Its representation, denoted SD, will be called the *believed system*. Finally, as written above, MBD aims at retrieving abnormalities in the system, abnormalities that are always dependent on the user viewpoint on the real system.

After introducing MBD foundations by relating the real-world system and its representation - the believed system -, let us briefly recall the classical model-based diagnosis framework that will be used throughout this paper [2] [1].

Definition 1 (Believed system)**.** *A believed system $\mathcal{S}$ is a pair $(\mathsf{SD},\mathsf{COMPS})$ where:*

1) *SD, the believed system description, is a set of first-order sentences.*
2) *COMPS, the believed system components, is a finite set of constants representing the real-world system physical units to diagnose* [2].

Observations are one of the few connections between real and believed systems. Intuitively, observations are captured *from the real system* by a set of *sensors* measuring the value $\mathsf{v}(\mathsf{p})$ of a real parameter p; and used, along with the believed system and p and $\mathsf{v}(\mathsf{p})$ representations, in the diagnostic reasoning.

Definition 2 (Observations)**.** *The set of observations, OBS, is a set of first-order sentences.*

Relying on the definitions of believed systems and observations, a diagnostic problem is the tuple $(\mathsf{SD},\mathsf{COMPS},\mathsf{OBS})$.

Diagnostic problems are solved according to the model-theoretical definition of diagnosis relying on the concept of believed system health state, $\sigma(\Delta,\mathsf{COMPS}\setminus\Delta) = [\bigwedge_{\mathsf{c}\in\Delta}\mathsf{Ab}(\mathsf{c})] \wedge [\bigwedge_{\mathsf{c}\in(\mathsf{COMPS}\setminus\Delta)}\neg\mathsf{Ab}(\mathsf{c})]$; where the predicate $\mathsf{Ab}(\mathsf{c})$ (resp. $\neg\mathsf{Ab}(\mathsf{c})$) represents the abnormality (resp. normality) of the component $\mathsf{c}\in\mathsf{COMPS}$.

Definition 3 (Diagnosis)**.** *Let $\Delta\subseteq\mathsf{COMPS}$. A diagnosis, D, for the diagnostic problem $(\mathsf{SD},\mathsf{COMPS},\mathsf{OBS})$ is the set of all diagnosis hypotheses $\sigma(\Delta,\mathsf{COMPS}\setminus\Delta)$ such that:*

$$\mathsf{SD}\cup\mathsf{OBS}\cup\sigma(\Delta,\mathsf{COMPS}\setminus\Delta)$$

is satisfiable.

[2]We assume that there is a bijection between R and COMPS, as R is defined by the user and the actions that can be performed on R if one of the unit is abnormal.

Finally, in real-world applications a diagnostic algorithm, noted $\mathcal{A}$, computes diagnoses ideally following the framework provided by Definition 3.

B. Characterising meta-diagnoses

Subsection II-A developed around diagnostic systems.

Definition 4 (Diagnostic system)**.** *A diagnostic system is a tuple $(\mathsf{SD},\mathsf{COMPS},\mathsf{OBS},\mathcal{A})$.*

Now, contrary to most classical hypotheses, diagnostic systems can be abnormal with respect to some properties one wants them to have. For example: modeling errors can result in false believed systems; observations can be different from the real parameter values due to perception errors; and diagnostic algorithms can produce diagnostic hypotheses not respecting the model-theoretic definition. Straightforwardly, meta-diagnosis aims at finding such abnormalities. In this subsection, based on [3], we provide a brief overview of meta-diagnosis.

First of all, the atoms in meta-diagnosis are *meta-components*, i.e. the parts of the diagnostic system whose normality/abnormality one wants to assess. Meta-components behavior and interactions are described in a *meta-system description* thanks to the unary predicate $\mathsf{M\text{-}Ab}(\cdot)$; which carries the semantic of meta-component's abnormality. Meta-components and meta-system descriptions form *meta-systems*:

Definition 5 (Meta-system)**.** *A meta-system is a pair $(\mathsf{M\text{-}SD},\mathsf{M\text{-}COMPS})$ where:*

1) *$\mathsf{M\text{-}SD}$, the meta-system description, is a set of first-order sentences.*
2) *$\mathsf{M\text{-}COMPS}$, the meta-system components, is a finite set of constants.*

The choice of meta-components depends on the goals and underlying hypotheses. As so, meta-components can be defined at different abstraction levels and it is not mandatory for every element of a diagnostic system to be considered a meta-component. For instance, if for a given problem one assumes that diagnostic algorithms and observations are never abnormal; to determine if each sentence in the believed system description describes the real system behaviour in a correct manner; one can associate a meta-component to every sentence in the believed system description and no meta-components to the rest of the diagnostic system.

From meta-systems one can move to introducing *meta-observations*. These can be, by and large, every possible observation at a diagnosis level along with every possible observation about the diagnostic system itself, such as the real-world system health state.

Definition 6 (Meta-observations)**.** *The set of meta-observations, $\mathsf{M\text{-}OBS}$, is a finite set of first-order sentences.*

Examples of meta-observations obtained from available test cases could be real system health state σ_{real}, the diagnoses computed by a diagnostic algorithm, observations and so on.

With meta-components, a meta-system description and some meta-observations; a meta-diagnostic problem is ready to be solved.

Definition 7 (Meta-diagnostic problem)**.** *A meta-diagnostic problem* $M\text{-}DP$ *is a tuple* $(M\text{-}SD,M\text{-}COMPS,M\text{-}OBS)$.

Solving a diagnostic problem consists in determining the normality/abnormality of meta-components. This is why, before focusing on a definition of meta-diagnoses, one needs to apprehend the concept of meta-health state:

Definition 8 (Meta-health state)**.** *Let* $\Phi \subseteq M\text{-}COMPS$ *be a set of meta-components. The meta-health state* $\pi(\Phi,M\text{-}COMPS\setminus\Phi)$ *is the conjunction:*

$$[\bigwedge_{mc\in\Phi} M\text{-}Ab(mc)]\wedge[\bigwedge_{mc\in(M\text{-}COMPS\setminus\Phi)} \neg M\text{-}Ab(mc)]$$

From the notion of meta-health state one can now move to solving the meta-diagnostic problem.

Definition 9 (Meta-diagnosis)**.** *Let* $\Phi \subseteq M\text{-}COMPS$*. A meta-diagnosis,* $M\text{-}D$*, for the meta-diagnostic problem* $(M\text{-}SD,M\text{-}COMPS,M\text{-}OBS)$ *is the set of all M-D hypotheses* $\pi(\Phi,M\text{-}COMPS\setminus\Phi)$ *such that:*

$$M\text{-}SD\cup M\text{-}OBS\cup\pi(\Phi,M\text{-}COMPS\setminus\Phi)$$

is satisfiable.

Finally, one shall notice that the diagnosis and meta-diagnosis worlds are not so different. In fact, if one takes a close look at Definitions 3 and 9 one may see that they are syntactically equivalent; the only difference being that a meta-diagnostic problem can be seen as a diagnostic problem where the artifact being diagnosed is a diagnostic system. A corollary is that every diagnostic algorithm can become a meta-diagnostic algorithm if it is sound and complete with respect to the underlying semantics [5]. This fact will prove extremely useful in Section III.

C. Diagnostic system and diagnostic results' properties and property relations

At this point one has a complete characterisation meta-diagnoses enabling the detection and isolation of abnormal properties in diagnostic systems. However, in order to provide the principles of MEDITO is Section III, we are still lacking of a description of some [3] diagnostic system and diagnostic results' properties and relations; which will be summarily provided, hereafter, based on [6] and [3].

1) Diagnostic result properties: Model-theoretic diagnoses' quality can be evaluated thanks to two properties: *validity* and *certainty*. Let us, in this paper, focus on nothing but the former.

Definition 10 (Validity of a diagnosis)**.** *Let* σ_{real} *be the believed system health state such that, for every* $c\in COMPS$*, if* c *is the image of* $r\in R$*: 1) if* r *is abnormal,* $\neg Ab(c)\wedge\sigma_{real}\models\perp$*; and 2) if* r *is normal,* $Ab(c)\wedge\sigma_{real}\models\perp$*. A diagnostic result,* D*, is said to be valid if* $\sigma_{real}\in D$*; and invalid otherwise.*

[3]In this paper we provide nothing but the diagnostic system and diagnostic result properties injected into the current version of MEDITO code. Nevertheless, MEDITO can be easily extended to account for other properties and relations such as the ones in [6] (cf. Section III).

As so, valid diagnoses are, intuitively, those covering the correct physical units to replace.

2) Observations properties: Concerning observations, we will focus on a single property, truth, assuring that real-world system parameter values are correctly perceived.

Definition 11 (Truth of the observations)**.** *Let* Ω *be the set of all structures and* $\Psi\in\Omega$ *the raw information about the reality. The observations* OBS *are an ontological truth iff* $\exists_{s\in Mod(OBS)}\ \exists_{t\in\Omega}\ (s\subseteq t)\wedge(t\rightleftharpoons\Psi)$.

3) Believed system properties: Among the various interesting believed system properties we will, as we did when describing observations properties (cf. Subsection II-C2), focus on a single property, truth, assuring that for every system description sentence stating X, X happens in reality.

Definition 12 (Truth of the believed system)**.** *Let* Ω *be the set of all structures and* $\Psi\in\Omega$ *the raw information about the reality. A believed system is an ontological truth iff, for all true* OBS*,* $\exists_{s\in Mod(SD\cup OBS)}\ \exists_{t\in\Omega}\ (s\subseteq t)\wedge(t\rightleftharpoons\Psi)$ *[7].*

4) Property relations: With the series of diagnostic system and diagnostic result properties described above, one can finally focus on an important relation between such properties (among the many others described in [6] and [3]); establishing that model-theoretic diagnoses obtained using an ontologically true believed system and some ontologically true observations are always valid:

Theorem 1. *If* $(SD,COMPS)$ *is an ontologically true believed system, then for every diagnostic problem* $(SD,COMPS,OBS)$ *with ontologically true observations, every model-theoretic diagnosis* $D_{M\text{-}T}$ *is valid.*

This theorem, whose proof can be found in [3], will become part of MEDITO's core in the following section.

III. MEDITO: A META-DIAGNOSIS TOOL

Up until this point the reader as been provided with both a diagnosis and a meta-diagnosis theory that, although far from a practical application, are a necessary condition for building one. It is now time to introduce MEDITO: a logic-based MEta-DIagnostic TOol aiming at computing the diagnostic systems's normal and abnormal meta-components.

As for the computational implementation, MEDITO relies on the fact that every meta-diagnostic problem can be seen as diagnostic one. This way, MEDITO uses Zhao and Ouyang [8] [9] diagnostic algorithms for determining all kernel meta-diagnoses for a given meta-diagnostic problem. Since these algorithms require a test for determining the consistency of a logical theory, MEDITO also implements a Constraint Satisfaction Problem (CSP) solver, *CHOCO* [10]. Note that the choice of CHOCO (as a CSP solver) and Zhao and Ouyang algorithms (as diagnostic algorithms) was one among the many possible choices of diagnostic algorithms and CSP solvers available.

With respect to MEDITO limitations, due to CHOCO language constraints, MEDITO can only meta-diagnose those

diagnostic systems where SD is a finite first-order theory; a reasonable limitation in real-world applications with no serious consequences.

In the two subsections that follow, we will focus on: 1) providing a general view on MEDITO's architecture, 2) giving a detailed view on MEDITO's modules.

A. MEDITO's architecture

Let one start by focusing on MEDITO's architecture.

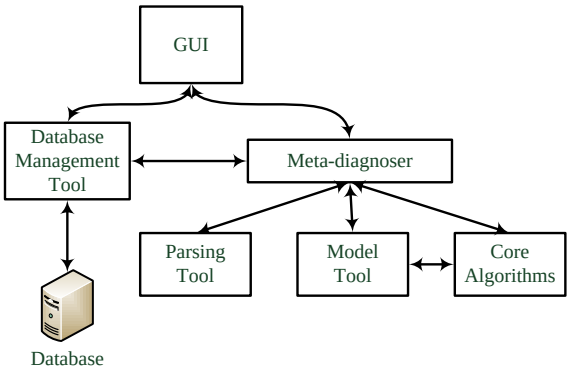

Figure 1. MEDITO's architecture

As depicted in Figure 1, MEDITO is composed of six interacting modules, where:

- the *GUI* module provides an Graphical User Interface for handling user inputs such as adding, editing or removing believed systems, observations and meta-observations; as well as for displaying the meta-diagnostic results,
- the *Database Management Tool* module is responsible for managing and querying MEDITO's SQL database containing believed systems, observations and meta-observations,
- the *Meta-diagnoser* module manages the computation of meta-diagnoses based on a user choice of meta-components and test-cases, i.e. pairs of observations and meta-observations,
- the *Parsing Tool* module transforms user-input text into CHOCO Java objects,
- the *Model Tool* module creates a CSP model in CHOCO and checks for the consistency between such model and a given CHOCO constraint, and
- the *Core Algorithms* module is responsible for implementing Zhao and Ouyang algorithms [8] [9] for computing all hitting sets and all minimal conflict sets for a given theory.

B. A detailed view on MEDITO's modules

Following the architectural overview of MEDITO in the preceding subsection, we provide, hereafter, a detailed view of each MEDITO module.

1) MEDITO's GUI module: First of all, MEDITO's Graphical User Interface is divided in three main screens.

In MEDITO's *System manager* screen, depicted in Figure 2, the user is able to create, change and delete a believed system. Believed systems are represented by two text blocs, the first one containing the believed system components and observables' declarations and the second one containing the system description.

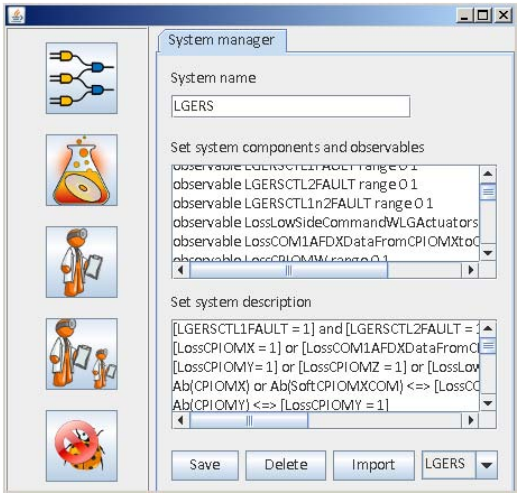

Figure 2. MEDITO's System Manager screen

In the first text bloc, text lines of the form:

- "component c", and
- "observable o range a b"

are used to declare, respectively, a believed system component c and a believed system observable o ranging from value a to b.

As for the second text bloc, a system description is declared through a series of text lines; each of them representing a SD-sentence in the Skolem normal form and with all the universal quantifiers omitted.

From MEDITO's *System manager* screen we can move to describing MEDITO's *Test Cases manager* screen where the user is able to create, change and delete a series of test-cases, i.e. pairs of observations and meta-observations, for each believed system. Observations and meta-observations are represented by two text blocs containing text lines representing OBS and M-OBS sentences.

Finally, MEDITO's *Meta Diagnoser* screen, (cf. Figure 3), enables the user to chose a believed system and a series of test-cases to compute meta-diagnoses, according to some preferences on meta-components. In MEDITO's current version, one can chose [4]:

- every SD-sentence to be a meta-component whose property of truth may be lacking,

[4]MEDITO's modularity enables easily adding new meta-component choices by changing the *GUI* module's *Meta Diagnoser* screen and the *Meta Diagnoser* module.

- every OBS-sentence to be a meta-component whose property of truth may be lacking, and/or
- the diagnostic algorithm to be a meta-components whose properties of soundness and/or completeness may be lacking [5].

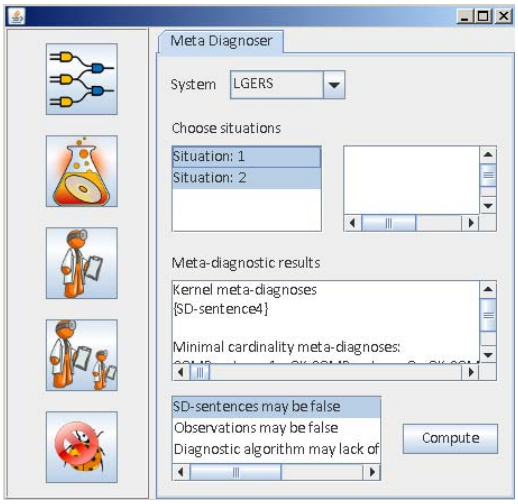

Figure 3. MEDITO's Meta-diagnoser screen

Finally, according to users choices, kernel meta-diagnoses are displayed in this same screen.

2) MEDITO's Database Management Tool module and database schema: As for the *Database Management Tool* (DMT) module, it enables the user to store, change and delete believed systems and test-cases in a SQL database; and provides the *Meta-diagnoser* module access to such database in order to compute meta-diagnoses. Figure 4 illustrates MEDITO's SQL database schema.

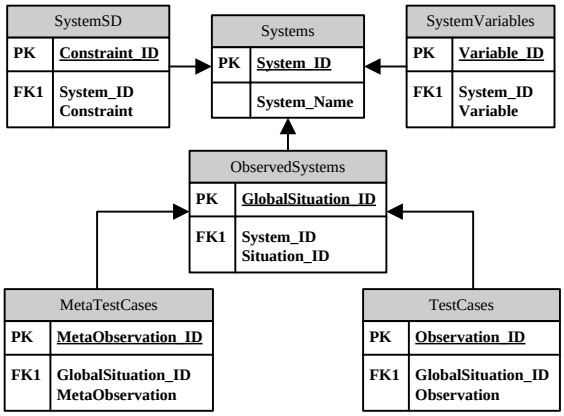

Figure 4. MEDITO's database schema

3) MEDITO's Meta-diagnoser module: The *Meta-diagnoser* module has the objective of managing the computation of meta-diagnoses by making use of the *Parsing Tool* (PT), *Model Tool* (MT) and *Core Algorithms* (CA) modules.

[5]In the present paper no details are given on this functionality since, by format constraints, the theoretical foundations needed for introducing it were not given in Section I.

The algorithm implemented in the *Meta-diagnoser* module is provided hereafter.

Input: A believed system name, systemName
Input: A test-case identification, situationID
Input: Meta-comps. choice, {sdOp,obsOp,algOp}

{Step1: Get observables (Ob), COMPS, OBS, M-OBS from DB, parse urser-text and initialize variables}
mdPbText ← DMT.imp(systemName,situationID)
pb ← PT.parse(mdPbText)
pb.mOb ← pb.Ob
pb.mSD ← ∅

{Step2: Create M-COMPS, M-SD and M-OBS based on user meta-component choices}
if sdOp = TRUE **then**
  pb.mCOMPS ← createMCOMPS(pb.SD)
  pb.mSD ← pb.mSD∪transfToMeta(pb.SD)
  pb.mOb ← pb.COMPS∪pb.mOb
end if
if obsOp = TRUE **then**
  pb.mCOMPS ← pb.mCOMPS ∪ createM-COMPS(pb.OBS)
  pb.mSD←pb.mSD∪sToMeta(pb.OBS)
  pb.mOb ← pb.COMPS∪pb.mOb
else
  pb.mOBS ← pb.mOBS∪pb.OBS
end if
if algOp = TRUE **then**
  (...)
end if

{Step3: Solve the meta-diagnostic problem}
CSPmod ← MT.create(pb)
(mDiag,mDiagMinCard) ← CA.metaDiag(CSPmod,pb.mCOMPS)
return (mDiag,mDiagMinCard)

Intuitively, this algorithm's:

Step 1: serves the purpose of importing a meta-diagnostic problem using *Database Management Tool* module's import function; using the *Parsing Tool* module to transform user-input-text components and observables declarations into CHOCO variables; and system descriptions, observations and meta-observations into CHOCO constraints.

Step 2: serves the purpose of building meta-components and a meta-system description, by taking user choices of meta-components into account.

Firstly, function createMCOMPS(·) associates a meta-component to each sentence in the input theory.

As for function transfToMeta(·), it transforms every SD-sentences and/or OBS-sentences stating "A" into meta-SD-sentences: ¬M-Ab(mc) ⇒ A; where mc is the meta-component associated to the sentence. This is as so because if a sentence stating "A" is not guaranteed to be ontologically true, then either "A" is ontologically true or the meta-

component mc associated to such sentence is abnormal, i.e. $\neg$M-Ab(mc) $\Rightarrow$ A.

Finally, if either the SD or OBS sentences can be ontologically false, the real-system health state is added as a possible meta-observable as a result of Theorem 1; and if the observations are assumed to be ontologically true, then they are added directly as meta-observations.

Step 3: uses the *Model Tool* and *Core ALgorithms* modules, respectively, to 1) create a CHOCO CSP model where the decision variables are the meta-components (though the function createModel($\cdot$)), and 2) compute, based on such model, all the kernel meta-diagnoses and minimal cardinality diagnoses for the meta-diagnostic problem (through the function metaDiagnose($\cdot$)).

4) MEDITO's Parsing Tool module: MEDITO's *Parsing Tool* module implements the parsing of user-input-text fields into CHOCO variables and constraints. This is done according to the syntax and semantics described in Subsubsection III-B1.

5) MEDITO's Model Tool module: The *Model Tool* module in MEDITO encodes two simple functionalities. First of all, it enables the construction of a CHOCO model based on a series of CHOCO variables and constraints. Secondly, it enables one to test the consistency between a CHOCO model and a constraint, by making use of CHOCO exception in the "propagate" method.

6) MEDITO's Core Algorithms module: The last module to be described is MEDITO's *Core Algorithms* module. By making use of the syntactic equivalence between diagnosis and meta-diagnosis (cf. Section II) this module implements Zhao and Ouyang diagnostic algorithms and applies them to meta-diagnosis to compute kernel meta-diagnoses. Moreover, it also transforms the problem into a min-CSP problem and solves it with CHOCO in order to obtain all the minimal cardinality meta-diagnoses. Finally, we state, once again, that this was one of the possible choices of algorithms.

In a more detailed manner:

- Zhao and Outang SE-tree based Minimal Hitting Set computation algorithm [8] is implemented in MEDITO's *Core Algorithms* module "computeMetaDiagnoses" function to compute the kernel meta-diagnoses for a given meta-diagnostic problem, based on all Minimal Conflict Sets for that same problem.
- In order to compute all the Minimal Conflict Sets for the meta-diagnostic problem, MEDITO's *Core Algorithms* module implements Zhao and Ouyang SE-tree based Minimal Conflict Set computation algorithm detailed in [9]. To do so, the algorithm takes as input the CHOCO model generated by the *Model Tool* module and representing the diagnostic problem to be solved, as well as the set of meta-components in the meta-diagnostic problem.
- Finally, Zhao and Ouyang algorithm for computing all Minimal Conflict Sets needs an underlying consistency checker to verify if a set is a conflict set. As so, MEDITO's *Core Algorithms* module transforms the set to be verified into a CHOCO conjunction constraint and makes use of *Model Tool* module's consistency check function to test the consistency between that constraint and the CHOCO model representing the meta-diagnostic problem.

As for the minimal cardinality meta-diagnoses, these are obtained by adding a constraint on the number of abnormal meta-components to the CHOCO CSP model representing the meta-diagnostic problem; and solving the resulting new CSP model by minimizing such constraint.

IV. Meta-diagnosing an Airbus landing gear extraction and retraction diagnostic system

Once a detailed description of MEDITO has been given in Section III, one is now able to put such a tool to work in a real-world application: an Airbus A380 Landing Gear Extraction and Retraction System (LGERS) diagnostic system. In order to do so, we will start this section by giving a general description of the real-world system; introduce a possible LGERS believed system used at Airbus; and meta-diagnose such believed system taking real-world observations and meta-observations into account.

A. An Airbus landing gear extraction and retraction system

The purpose of the Airbus A380 Landing Gear Extension and Retraction System (LGERS) is to provide controlled Landing Gear extension and retraction and door opening/closing such that the aircraft can perform a normal landing and cruise flight.

Structurally speaking, an A380 landing gear consists of: a retracting Nose Landing Gear (NLG), two retracting Wing Landing Gears (WLG), and two retracting Body Landing Gears (BLG).

In terms of the associated actuators, all the landing gears (NLG, WLG and BLG) have retract, unlock and gear uplock actuators; and the associated doors are, for the most part, hydraulically operated through some hydraulic actuators, being the remaining doors mechanically operated.

As for the Landing Gear Control [and Indication] System (LGCIS), its simplified general architecture is depicted in Figure 5, where component real names and some details have been changed/removed for confidentiality issues.

First of all, one shall note that LGCIS is decomposed, for safety purposes, into two redundant sides executing, by and large, the same functions in alternate active/standby modes.

In terms of sensor data, each LGCIS side has its own sensor inputs which are transmitted to the Core Processing Input and Output Modules (CPIOM) - that can be seen as aircraft generic computers - through some Remote Data Concentrators (RDC) by means of Arinc 429 data links.

Using such sensor data and the control commands provided by the landing gear control lever (not represented

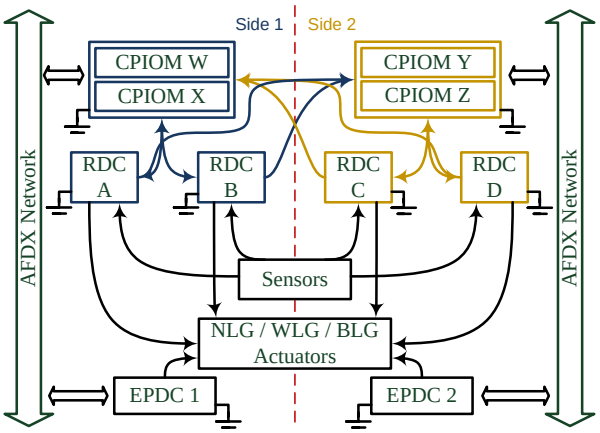

Figure 5. LGCIS architecture

in Figure 5), CPIOM W (resp. Y) LGERS COM HIGH software computes and sends control signals via the AFDX network to the Electrical Power Distribution Centre 1 (resp. 2) controlling the actuators; and CPIOM X (resp. Z) LGERS COM LOW software computes and sends control signals through RDCs A and B (resp. C and D) to the actuators.

B. An Airbus LGERS believed system

From the brief description of an Airbus A380 LGERS given in the previous subsection, let us move to providing the reader with a very brief description of Airbus diagnostic system and introduce a possible believed system representing the presented A380 LGERS.

Due to its complexity, airbus aircraft rely on a semi-distributed diagnostic architecture with a series of distributed system-level diagnostic agents, called Built-In Test Equipments (BITE); and a central diagnoser, called Centralised Maintenance System (CMS), responsible for processing information from BITEs and from the Flight Warning System (FWS) and computing an aircraft diagnosis.

As for BITE, these pieces of software detect and isolate system-level failures where maintenance is needed, based on some discrepancies between a system's normal and current behavior; and send such maintenance information to the CMS. An example of such failures is the loss of control signal information given by the CPIOM X to the CPIOM W through the AFDX network, represented as "Loss of COM1 AFDX data from CPIOM X to CPIOM W".

The FWS, on its turn, determines if any aircraft-level failure with an operational impact is present; based on all the detected discrepancies in the aircraft, as well as on the aircraft operating conditions. An example of such failures is the loss of all the LGCIS side 1 function, represented as "L/G CTL 1 FAULT". If this is the case, it sends this failure information to the aircraft crew and to the CMS by means of warnings.

Based on BITE and FWS information, as well as on a representation of each aircraft system and their interactions, the CMS computes an aircraft-level diagnosis.

Now, relying on each system description, such as the LGERS one provided in Subsection IV-A, and on BITE/FWS isolation capabilities, Airbus engineers build a CMS aircraft believed system. Suppose that a part of such believed system is depicted in Figure 6, where: 1) continuous arrows symbolize material implications, 2) dashed arrows symbolize unrepresented nodes present in the believed system, and 3) there is an underlying hypothesis that only the predecessors of a node imply it (completeness hypothesis).

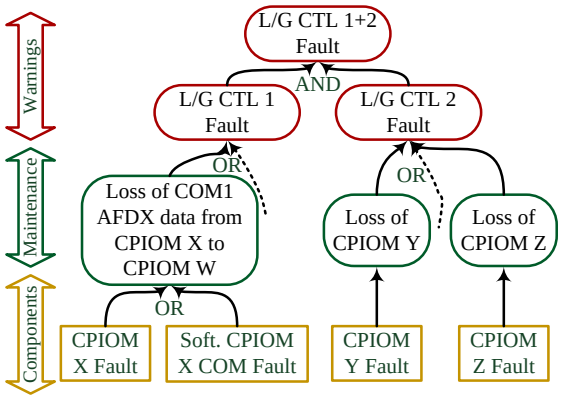

Figure 6. LGCIS model used by the diagnostic system

So, for instance, the nodes L/G CTL 1 Fault, L/G CTL 1 Fault and L/G CTL 1+2 Fault as well as their edge relations are involved in the first-order sentence: [L/G CTL 1 Fault = $\top$] $\wedge$ [L/G CTL 2 Fault = $\top$] $\Leftrightarrow$ [L/G CTL 1+2 Fault = $\top$]; where the reader may notice the equivalence relation due to the completeness hypothesis [6].

Finally, to provide the reader with some intuition on such model, we can see, for instance, that engineers expect LGCIS side 2 to be lost after a CPIOM Y fault, since as described before the LGERS COM HIGH software in this CPIOM computes non-redundant side 2 control inputs.

C. Meta-diagnosing an Airbus LGERS believed system with MEDITO

As mentioned before, reality is sometimes different from what one expects; and engineers continuously struggle to produce ontologically true believed systems - a necessary (although insufficient) condition to assure correct component replacements (cf. Theorem 1). In this Subsection, we provide an example of how to use MEDITO to automatically detect and isolate ontologically false sentences in LGERS SD.

Using MEDITO's "System manager" screen, the first thing to do is to provide MEDITO with the LGERS believed system one wants to meta-diagnose such as the one depicted in Figure 6 (cf. Figure 2) [7].

Once this is done, observations and meta-observations are added in MEDITO's "test Cases manager" screen. In this

[6] $\top$ represents logical true; and $\bot$ represents logical false.

[7] For space constraints the full LGERS believed system is not presented.

example, a test-case was added, by using some [not] received FWS warnings: OBS = {LGERSCTL1n2FAULT=$\bot$, LGERSCTL1FAULT=$\top$, LGERSCTL2FAULT=$\bot$}; and some mechanic feedback on real-world system unit faults:M-OBS = {Ab(CPIOMW), $\neg$Ab(CPIOMX), $\neg$Ab(CPIOMZ), $\neg$Ab(SoftCPIOMXCOM)}.

Finally, in the "Meta Diagnoser" screen, the option "SD-sentences may be false" is selected.

In this example, MEDITO computed three kernel diagnoses for a fifteen-sentences SD in 29689 milliseconds; as well as three minimal cardinality diagnoses in 63 miliseconds [8]. The computed kernel meta-diagnoses were:

{SD-sentence2}, {SD-sentence8}, {SD-sentence4}

where:

SD-sentence2: [LossCPIOMX=$\top$] $\vee$ [LossCOM1AFDX DataFromXtoW=$\top$] $\vee$ ... $\Leftrightarrow$ [LGERSCTL1FAULT=$\top$]

SD-sentence4: Ab(CPIOMX) $\vee$ Ab(SoftCPIOM XCOM) $\Leftrightarrow$ [LossCOM1AFDXDataFromXtoW=$\top$]

SD-sentence8: Ab(CPIOMX)$\Leftrightarrow$[LossCPIOMX=$\top$]

Based on these results, one can see that either SD-sentence2, SD-sentence4 or SD-sentence8 are ontologically false. Now, as proved in [3] meta-diagnosis reasoning is monotonic (assuming the meta-observations are always true) and, being our meta-diagnosis engine sound and complete with respect to the underlying logic, adding new meta-observations can only reduce the set of meta-diagnoses.

As so, the provided test-case was extended by adding some received BITE information to the set OBS: LossCOM1AFDXDataFromXtoW=$\bot$. As depicted in Figure 3, the computed kernel meta-diagnoses were:

{SD-sentence4}

The fact that {SD-sentence4} was ontologically false was later found: the abnormality of CPIOM W can also lead to the loss of COM 1 AFDX data from CPIOM X to CPIOM W; since the later CPIOM cannot acknowledge receiving the data. This was confirmed by changing SD-sentence4 to Ab(CPIOMX) $\vee$ Ab(SoftCPIOMXCOM) $\vee$ Ab(CPIOMW) $\Leftrightarrow$ [LossCOM1AFDXDataFromXtoW=$\top$]; and noticing the resulting empty set of kernel meta-diagnoses.

V. Discussion

Based on the frameworks of diagnosis and meta-diagnosis of [1], [2] and [3] we have offered a detailed architecture and functional description of MEDITO: a logic-based meta-diagnostic tool. Relying on Zhao and Ouyung diagnostic algorithms [8] [9] and on CHOCO CSP solver [10], MEDITO provides empirical proof supporting meta-diagnosis claim that any diagnostic algorithm can be used to solve a meta-diagnostic problem.

Moreover, using MEDITO, we have meta-diagnosed an Airbus A380 LGERS believed system with very positive practical results. The success of this practical application is currently letting Airbus engineers envisage using MEDITO to meta-diagnose the whole aircraft believed system by relying on in-flight observations and meta-observations.

As for its kernel meta-diagnosis computational performances, by relying on Zhao and Ouyang diagnostic algorithms MEDITO directly inherits from these algorithms worst case complexity of $O(2^n)$ and consequent inability to handle large amounts of meta-components (as it is the case with every diagnostic algorithm aiming at computing every diagnosis we are aware of). Moreover, MEDITO's performance is proportional to the time needed by CHOCO to execute the consistency check of a CSP model.

Kernel meta-diagnosis somehow poor computational performances were handled by compromising finding all the meta-diagnoses and computing only the minimal cardinality ones. This, in turn, can be seen as a min-CSP problem and handled in CHOCO with much better performances.

All in all, meta-diagnosis complexity can be handled in MEDITO by making use of classical Model-Based Diagnosis techniques such as hierarchical diagnosis; and by optimizing the CSP model constraint choice. This will will be a part of our future works.

[8]Intel Core 2 Duo CPU at 2.4GHz and 2Gb of RAM

References

[1] R. Reiter, "A theory of diagnosis from first principles," *Artif. Intell.*, vol. 32, no. 1, 1987.

[2] J. de Kleer and B. C. Williams, "Diagnosing multiple faults," *Artif. Intell.*, vol. 32, no. 1, 1987.

[3] N. Belard, Y. Pencolé, and M. Combacau, "A theory of meta-diagnosis: Reasoning about diagnostic systems," in *Proc. IJCAI-11 22nd International Joint Conference on Artificial Intelligence*, in press.

[4] G. S. Ravi Kapadia and M. Walker, "Real world model-based fault management," in *Proc. DX-07 18th International Workshop on Principles of Diagnosis*, 2007.

[5] W. Hodges, *Model Theory*, ser. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1993.

[6] N. Belard, Y. Pencolé, and M. Combacau, "Defining and exploring properties in diagnostic systems," in *Proc. DX-10 21th Int. Workshop on Principles of Diagnosis*, 2010.

[7] A. Tarski, "The concept of truth in formalized languages," in *Logic, Semantics, Metamathematics*. Oxford: Oxford University Press, 1936.

[8] X. Zhao and D. Ouyang, "A method of combining se-tree to compute all minimal hitting sets," *Progress in Natural Science*, vol. 16, pp. 169–174, 2006.

[9] ——, "Improved algorithms for deriving all minimal conflict sets in model-based diagnosis," in *Proc. of the 3rd ICIC*, 2007.

[10] C. Team, "choco: an open source java constraint programming library," Ecole des Mines de Nantes, Research report, 2010. [Online]. Available: http://www.emn.fr/z-info/choco-solver/pdf/choco-presentation.pdf