Advances in Intelligent Health Reasoning and its Application to IBDM

Alexander Feldman,[1,2] Marco Caporicci,[3] Oscar Gracia,[3] and André Bos [1]
[1]Science & Technology BV, P.O. Box 608, 2600 AP, Delft, The Netherlands
Tel.: +31 15 2629889, Fax: +31 15 2629567, e-mail: {feldman,bos}@science-and-technology.nl
[2]Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science
Mekelweg 4, 2628 CD, Delft, The Netherlands
Tel.: +31 15 2781935, Fax: +31 15 2786632
[3]Reentry and Human Transportation Division, Development Department D/HME, ESTEC
P.O. Box 299, 2200 AG Noordwijk, The Netherlands
Tel.: +31 71 5656065, Fax: +31 71 5655754, e-mail: {marco.caporicci,oscar.gracia}@esa.int

Abstract—We report on an experiment of using Model-Based Diagnosis (MBD) for advanced health monitoring of the hard-docking system of the International Berthing and Docking Mechanism (IBDM). The results of this experiment illustrate our approach in solving two major problems of MBD: modeling an artifact of non-trivial size and managing the representation complexity for finding a root cause of failure. The qualitative model of the hard-docking system has 425 variables (161 components) and 1328 clauses wherein the diagnostic reasoner is capable of determining a quadruple failure in less than 6 s on a midrange PC. Our conclusion is that the qualitative modeling language LYDIA and the automated reasoning framework UPTIME are capable of delivering of sound and accurate diagnosis imposing only a moderate computational burden, making this technology attractive for future health monitoring applications.

TABLE OF CONTENTS

1. INTRODUCTION

Motivated by the success of NASA in achieving autonomy by using Model-Based Reasoning (MBR) and automated planning in the inter-planetary probe Deep Space 1 [16], we have studied the application of Model-Based Diagnosis (MBD) techniques for achieving accurate and fast determination of root-cause of failure to a wider-range of critical and dependent systems. This paper reports on an experiment of using MBR for advanced health monitoring of the hard-docking system of the International Berthing and Docking Mechanism (IBDM).

In our results we restate the conclusions of related research [16], [19], [25] that MBD delivers a robust and accurate determination of root-cause of failure and the technique is better scalable in comparison to traditional procedure-based fault monitoring. The major disadvantage of the traditional techniques is the additional fault modeling effort that is required to express the fault behavior of the system. It requires that a human manually simulates the fault propagation: a labor-intensive and error-prone process. Furthermore this process of describing the faulty-behavior of each component of a system needs to be repeated each time the design changes, where in the model-based approach, the modeling effort can be restricted to the structural and nominal behavior of the system; a process that can be often automated for a substantial part.

MBD, on the other hand, does not come without its difficulties and together with the description of our approach we propose a number of techniques to deal with these challenges. We consider two main aspects of using MBD for the health-monitoring of IBDM - modeling and computational [24]. In the former we describe the process of building a model from the system design specification. In the computational part, we introduce a number of techniques for controlling the combinatorial explosion when computing diagnosis.

The IBDM hard-docking system is a complex device comprising of tens of interconnected electro-mechanical components. Modeling and reasoning about a device of such size and character is non-trivial and requires a number of language features for fault modeling and the use of advanced algorithms, which we have implemented as a part of the UPTIME MBR integration platform.

In the modeling phase of the hard-docking system we have used a specialized language for qualitative modeling, named LYDIA (Language for sYstem DIAgnosis) [17]. In addition to demonstrating the language use, we motivate the modeling approximations and describe the process of deriving and refining the model. We pay special attention to the temporal aspects of diagnosis.

1-4244-0525-4/07/$20.00

On our way of solving the big computational burden of MBD we employ a number of state-of-the art techniques like model compilation, multi-valued reasoning, conflict-directed A* search, exploitation of structure (hierarchy) and others. The IBDM model, due to its highly structured nature, proves to be very amenable to these techniques, delivering fast diagnosis time.

Almost all approaches towards the problem of fault detection of technical systems check whether the observed sensor readings agree with some predefined nominal behavior. For simple systems, nominal behavior can often be described in a simple way, by defining static upper and lower limit values for key system variables. For complex systems, however, defining static limit values in this way quickly fails short as normative behavior will be state dependent. The fault detection approaches for these complex systems need a system tracking or a state estimation process in order to derive the appropriate limit values to define nominal behavior. Typical approaches for system tracking or state estimation include simulation (e.g., discrete simulation using state transition formalisms [23]), filters (e.g., Kalman filters [1]), and rule-based systems [18].

Traditional techniques for fault diagnosis often use a form of pattern matching to derive the root cause of a failure. A typical example is a fault propagation graph describing how fault behavior propagates through the system. This graph can then be applied to derive the cause of a failure by matching the observed systems to the fault behavior as described by the graph.

One of the most influential theoretical sources in Model-Based Diagnosis are [9] and [22]. An early implementation of these fundamental results is the General Diagnostic Engine (GDE), using a sound but incomplete consistency checking (Logic-Based Truth Maintenance System). Both LYDIA and UPTIME use DPLL-based SAT solvers [6] as basic inference engines. An important difference is that UPTIME provides a multi-valued reasoning mechanism [12], thus obsoleting the need of costly encodings [13]. Such kind of multi-valued reasoning can be seen a structure-exploitation for faster reasoning. This is complementary to conflict-driven backjumping and learning [26], a feature implemented in UPTIME as well.

Another approach for solving the high-computational cost of MBR is to use compilation. Compilation techniques trade inexpensive off-line preprocessing for faster reasoning at runtime. Strict compilation techniques [3], [4] achieve online reasoning time polynomial of the size of the compiled model representation. The biggest problem of the strict compilation approaches is that size of a compiled model can be (in the worst-case) exponential of the original representation, thus imposing limits on the applicability of these techniques.

The remainder of this paper is organized as following. In the next chapter we introduce the IBDM hard-docking system and the language for qualitative modeling LYDIA. In Section 3 we discuss some algorithms for automated reasoning. An extensive set of experiments is shown in Section 4 and finally, in Section 5, we summarize our research and outline directions for future work.

2. MODELING OF THE IBDM HARD-DOCKING SYSTEM

This section discusses the modeling aspects of the IBDM hard-docking system. We start with a description of the system being modeled. Afterwards, we illustrate all the steps necessary for creating a qualitative fault-model. With this running example we introduce the LYDIA language. In practice we have followed a mixed top-down and bottom-up approach in which we have firstly defined the system structure at top-level, then modeled each component separately and finally validated the overall system.

Overview of the IBDM Hard-Docking System

In reality the IBDM Hard-Docking System (Figure 1) is a fairly complex electro-mechanical device which establishes a rigid connection between the chaser and target vehicles once the soft-docking phase has been completed [11]. It is of interest for automated health-monitoring due to the complexity of the docking operation, the risk of critical or non-critical failures and the need for fast troubleshooting and decision making in case of a failure. The design of the hard-docking system is discussed briefly in the text below.

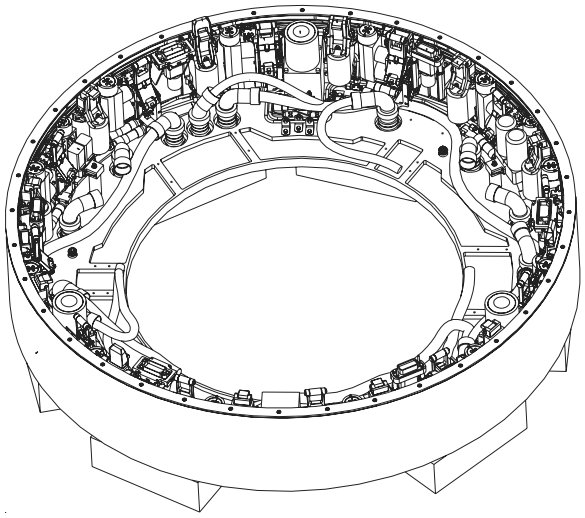

Figure 1. IBDM hard-docking system

The IBDM hard-docking system is being designed for robustness, hence the redundancy in its mechanisms. At the structural part we distinguish the following major subsystems:

EMA: The primary Electro-Mechanical Actuator (EMA) unit drives the primary ring gear. In its turn the primary ring gear drives the input of the twelve primary gear-boxes. Similarly, the secondary EMA unit drives the secondary ring gear that

drives the inputs of the twelve secondary gear-boxes.
Gear-Box: The gear-box (dual right angle gear head) has both the primary and secondary gear train built in. The box is attached directly to the latch frame. The gear-box is driven by the ring gears, using a separate pinion for each drive, and interfaces with the primary and secondary crank of the latch assembly by means of a spline.
Latch Assembly: The hard docking system consist of the tunnel that makes the pressurized connection between the two spacecrafts. The tunnel itself is not a part of the model. Twelve structural latches are mounted on the on the inside of the tunnel. The counter part of each latch assembly is a latch tab.
Latch Tab: The latch tab provides the interface for the latch linkage on the mating tunnel. The latch tab assembly is designed to act as a load limiter that will keep the forces in the latch within a defined range (due to manufacturing and assembly tolerances). The load limiter is built as a preloaded stack of Belleville springs, placed in series with the latch tab. This load limiter is equipped with a load sensing bolt to monitor the preload while the vehicles are mated in orbit.

The qualitative model of the hard-docking system is a set of constraints over some observable, health, and internal variables (a more formal definition of a health-model follows in Section 3). We group these constraints, depending on the discrete control state in which the system is. The values for the control state are human-derived; a small number to prevent computational blow-up when applying automated reasoning and enough to extract useful diagnostic information during the whole operational lifetime of the hard-docking system. A state transition diagram illustrating the possible transitions between the states is shown in Figure 2.

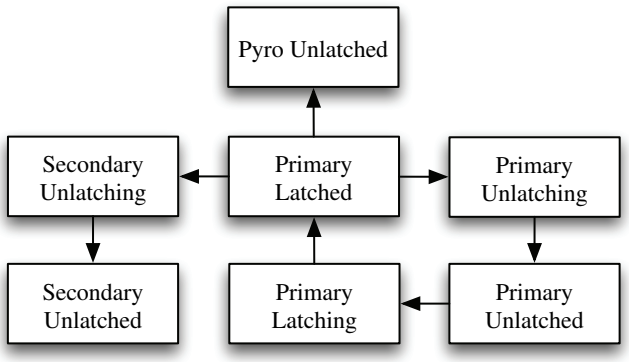

Figure 2. IBDM hard-docking system state transition diagram

There are seven phases in which we describe the behavior of the hard-docking system:

Primary Latching: The mechanical components are moving – the primary EMA is rotating the drive gear which moves the primary gear-boxes, which in their turn rotate the primary cranks of the latch assemblies. The rollers are engaging the corresponding latch tabs and the load on the load-sensing bolts is increasing. The state of the micro-switches in the latch-assemblies is irrelevant to the health of the system (i.e. they can be either open or closed depending on the progress of the latching).
Primary Latched: The hard-docking system can be put in a latched state by the primary mechanism only. The rollers engage the latch tabs. The load in the latch tabs should be constant in time and above a landmark-value. In the nominal (all-healthy) state, the micro-switches should reflect the closed state of the primary latch assembly.
Primary Unlatching: The opposite of the primary latching phase in which the rollers are retracting and the load in the latch tab is decreasing. The state of the micro-switches is, again, not relevant for the health of the system.
Secondary Unlatching: A back-up unlatching has been initiated and this should be reflected in the load of the latch tabs. A healthy secondary unlatching is indicated by a decreasing load in the load-sensing bolt.
Pyro Unlatch: The pyro unlatching system is provided for rapid-decoupling of the docked vessels in case of a catastrophic failure. Due to the nature of this subsystem (manually activated and being active for a very small time interval) we do not include it in the model.
Primary Unlatched: If the hard-docking system is in this state, we indicate the configuration in which the latch assembly was unlatched by the primary mechanisms.
Secondary Unlatched: The hard-docking system is in this state after a successful unlatching performed by the secondary mechanism (secondary EMA, secondary gear-box and secondary latch assembly).

Next we continue with the steps necessary for creating the model of the IBDM hard-docking system.

Modeling of the IBDM Hard-Docking System in LYDIA

A LYDIA model is a hypergraph describing the system structure, its subsystems and components. Each component defines a set of variables in the finite-integer domain. In LYDIA these variables can assume one or more symbol values (the notion of symbol is similar to the one in LISP). The reasoners, of course, work with integer sets while the symbols are for convenience. LYDIA is a *declarative* language with syntax similar to Verilog and an extensive typing system.

For brevity, we will not discuss the model of the IBDM hard-docking system in its entirety, but only these parts of it which fully illustrate the modeling process. We start with defining the domains of our variables. Next we model the top-level system, underlining some important modeling constructs (e.g., composition and constraints). Finally for this section, we discuss a single component (latch tab) with nontrivial temporal model. The models of the remaining components are built in similar way.

Defining Domain Types of the Model Variables— There are two atomic types in LYDIA: Boolean and an enumeration

(the former is for convenience only). In addition to that, LYDIA supports a feature-rich system of user-defined types like structures, arrays, aliases, etc. All these complex data-types are rewritten to atomic variables of enumerative type in the model compilation phase. We start the model of the hard-docking system by defining some user-types (cf. Figure 3).

```
type range = enum { below, inside, above };
type phase = enum
{
    primaryLatching,
    primaryUnlatching,
    secondaryUnlatching,
    pyroUnlatch,
    primaryLatched,
    primaryUnlatched,
    secondaryUnlatched
};
type roller = enum {engaging, engaged, disengaging, disengaged };
type systemControlState = enum
{
    healthy,
    spuriousLatched,
    spuriousUnlatched,
    inadvertentAction
};
```

Figure 3. User-defined types of the model of the IBDM hard-docking system.

Note the use of the enumeration type *phase* to denote the control state in which the hard-docking system functions. In LYDIA and UPTIME, there is no difference between health and nominal states. In place, an extensive mechanism for attributing variables has been developed, which we have used for assigning *a priori* probabilities of a variable assuming a given value. This probability will be used to solve an constraint optimization problem, i.e., finding such a health assignment which maximizes a utility function. The mechanism is then used for ordering the health states[1].

Top-Level System— Figure 4 and Figure 5 show part of the top-level node of the IBDM hard-docking system. The *range* enumeration is used in a fashion similar to [25] for denoting landmark intervals. The state of the hard-docking system (cf. Figure 2) is one of the values in the *phase* enumeration. Finally, we have four possible states for the roller which is a part of the latch assembly.

The "switch" predicate of Figure 4 splits the constraints depending on the control phase, the latter observed in the *dockingPhase* variable. Note that only the primary mechanism can be used for latching. LYDIA can use existential and universal quantifiers over the elements of arrays. We use this feature to implement constraints like the ones in Figure 4. The second **exists** construct, for example, specifies that an increased amount of force exercised by the ring gear implies any of the twelve gear boxes or latch tabs jammed.

[1] In some cases the ordering coincides with an ordering according the fault-cardinality of a state [7].

```
system HardDockingSystem()
{
    phase realDockingPhase, controlDockingPhase, dockingPhase;

    if (controlDockingPhase != dockingPhase) {
        controlState = systemControlState.inadvertentAction;
    }

    systemControlState controlState;

    if (systemControlState.healthy = controlState) {
        realDockingPhase = dockingPhase;
    }
    if (systemControlState.spuriousLatched = controlState) {
        (realDockingPhase = phase.primaryUnlatched) or
        (realDockingPhase = phase.secondaryUnlatched);
    }
    if (systemControlState.spuriousUnlatched = controlState) {
        (realDockingPhase = phase.primaryLatched);
    }

    switch (realDockingPhase) {
        phase.primaryLatching ->
        {
            primaryEMARegime = EMARegime.latching;
            latchTabPhase = tabPhase.latching;
        }
        phase.primaryUnlatching ->
        {
            primaryEMARegime = EMARegime.unlatching;
                        ⋮
    }

    tabPhase latchTabPhase;

    system LatchTab latchTab[12](latchTabPhase);

    EMARegime primaryEMARegime;
    EMARegime secondaryEMARegime;

    EMAMotorSelection primaryEMAMotorSelector;
    EMAMotorSelection secondaryEMAMotorSelector;
    system EMA primaryEMA(primaryEMAMotorSelector,
                          primaryEMARegime,
                          primaryRingGear);
    system EMA secondaryEMA(secondaryEMAMotorSelector,
                            secondaryEMARegime,
                            secondaryRingGear);
```

(continues in Figure 5)

Figure 4. A LYDIA model of the top level hard-docking system.

In Figure 4 the reader can observe two major LYDIA constructs – variable and subsystem declarations and variable constraints. For example, two variables of type *EMARegime* are used to denote the modes of the primary and secondary EMA components. The part of the model shown in Figure 4 contains the declaration of the twelve latch tab components (the model of the latch tab we will discuss in details below).

The other type of construct is a constraint. In Figure 4 we see mainly implications (the "if" and "switch" predicates are translated to multi-valued propositional implications as we will see in more details in Section 3). Note, that unlike in, e.g. Prolog, LYDIA does not impose tractability on the modeling language. The price of this is exponential complexity in

(continued from Figure 4)

```
    ringGear primaryRingGear, secondaryRingGear;

    system GearBox gearBox[12];
    system StructuralLatchAssembly structuralLatchAssembly[12];

    crank latchAssemblyPrimaryCrank[12];
    crank latchAssemblySecondaryCrank[12];
    roller latchRoller[12];

    gearBoxState primaryGearBoxState[12];
    gearBoxState secondaryGearBoxState[12];

    latchState primaryLatchState[12];
    latchState secondaryLatchState[12];

    forall (i in 0 .. 11) {
        gearBox[i](primaryGearBoxState[i],
                   secondaryGearBoxState[i],
                   primaryRingGear,
                   secondaryRingGear,
                   latchAssemblyPrimaryCrank[i],
                   latchAssemblySecondaryCrank[i]);
        structuralLatchAssembly[i](primaryLatchState[i],
                                   secondaryLatchState[i],
                                   latchAssemblyPrimaryCrank[i],
                                   latchAssemblySecondaryCrank[i],
                                   latchRoller[i]);
    }

    if ((primaryRingGear = ringGear.forceLatching) or
        (primaryRingGear = ringGear.forceUnlatching)) {
        exists (i in 0 .. 11) {
            (primaryGearBoxState[i] = gearBoxState.jammed) or
            (primaryLatchState[i] = latchState.jammed);
        }
    }

    if (secondaryRingGear = ringGear.forceUnlatching) {
        exists (i in 0 .. 11) {
            (secondaryGearBoxState[i] = gearBoxState.jammed) or
            (secondaryLatchState[i] = latchState.jammed);
        }
    }
}
```

Figure 5. A LYDIA model of the top level hard-docking system.

the reasoning, but this worst-case scenario does not appear in practice, something we will see in the experimental section of this paper.

Latch Tab— The latch tab is the counterpart of the latch assembly (the docking system is androgynous). It is supplied with a load-sensing bolt, which provides an important diagnostic information about the docking process. Note that we model the hard-docking system of the chaser vehicle only. Had it been assumed the diagnostic data to be observable on both the target and chaser vehicles, then it would be possible to discover a wider-range of fault situations[2]. In the current case, it is not possible to distinguish between, e.g., a broken latch tab and a broken roller in the opposite latch assembly in the target docking system. The model of the latch tab is shown in Figure 6.

[2] Such a combination of sensed data, however, would require a communication between the target and chaser diagnostic engines.

```
type tabState = enum { healthy, faulty };
type threshold = enum { below, above };
type sig = enum { minus, zero, plus };

type loadSensingBolt = struct { threshold load, sig fderSign };

type tabPhase = enum { latching, latched, unlatching, unlatched };

system LatchTab(tabPhase dockingTabPhase)
{
    loadSensingBolt bolt;
    attribute observable(bolt) = true;

    tabState state;
    attribute health(state) = true;
    attribute probability(state) = cond(state)
    (
        tabState.healthy -> 0.99;
        tabState.faulty -> 0.01
    );

    if (state = tabState.healthy) {
        switch (dockingTabPhase) {
            tabPhase.latching ->
            {
                bolt.fderSign = sig.plus;
            }
            tabPhase.latched ->
            {
                bolt.load = threshold.above;
                bolt.fderSign = sig.zero;
            }
            tabPhase.unlatching ->
            {
                bolt.fderSign = sig.minus;
            }
            tabPhase.unlatched ->
            {
                bolt.load = threshold.below;
                bolt.fderSign = sig.zero;
            }
        }
    }
}
```

Figure 6. A LYDIA model of the latch tab component.

The model of the latch tab uses a qualitative *landmark* value [14] to indicate the load on the bolt. It is represented as an enumeration of type *range*. In further implementation steps, there is a process which binds quantitative values to each qualitative type (e.g., it would be determined that a qualitative variable x of type *threshold* has the value x = *threshold.above* iff x > k, where k is the specified measurement of force (k represents the sensor reading).

The mechanism for deriving these bindings is outside the scope of this overview paper, but the mentioning of their use is essential for understanding the technique. The threshold value in this case distinguishes the latched from the unlatched stages. The load and time derivative of the load are combined in the LYDIA structure *loadSensingBolt* describing the load sensing bolt.

To model the load-sensing bolt, part of the latch tab, we use the *loadSensingBolt* structure. One of its variables (*fderSign*) contains the sign of $\frac{dF}{dt}$, where F is the force measured by

the load-sensing bolt. Such an explicit sign algebra provides a simple way of reasoning over (explicit) time and state. At present, LYDIA and UPTIME can not perform qualitative differentiation and integration but we are planning to implement such features for the future versions of the software and the IBDM model.

For this particular component we have specified a weak-fault model; that is the component is constrained in its healthy state only. The faulty behavior of the latch tab is not specified in the model. One of the implications of this is the derivation by the automatic reasoner of the conclusion that the component is faulty but it behaves is healthy. This is not a problem as the latter health-state has very low probability and the diagnosis search is terminated before such diagnoses have been generated. On the other hand, weak-fault modeling decreases the complexity of the reasoning [8].

3. AUTOMATED COMPUTATION OF DIAGNOSIS WITH LYDIA AND UPTIME

This section introduces some formalism for MBD and illustrates a couple of algorithms for computing diagnosis. We analyze compilation trade-offs between time and space with different representation.

Preliminaries

We start by briefly recalling the basic notation of consistency-based diagnosis [8].

Definition 1 (Diagnostic Problem). A diagnostic problem *DP* is defined as the triple $DP = \langle SD, COMPS, OBS \rangle$, where *SD* is a propositional theory describing the behavior of the system, *COMPS* is a set of assumable variables in *SD*, and *OBS* is a term stating an observation over some set of "measurable" variables in *SD*.

In this paper we assume that for any propositional theory *SD*, $SD \not\models$ and *SD* is not valid.

Intuitively, LYDIA compiles a model like the one discussed in Section 2 to *SD*, where *SD* is a language over which we can execute queries like, e.g., determining if $\Delta \not\models SD \quad OBS$.

Definition 2 (Diagnosis). A *diagnosis* for the system $DP = \langle SD, COMPS, OBS \rangle$ is a set $D \quad COMPS$ such that:

$$SD \quad OBS \qquad \neg h_c \qquad\qquad h_c \not\models$$
$$c \in D \qquad\qquad c \in (COMPS \setminus D)$$

When many variables in *OBS* are of uncertain values, which is often the case in practice, the diagnostic problem *DP* becomes underconstrained, leading to a large number of diagnoses (in the worst-case exponential of the number of components). Hence, we are interested in competing these diagnoses only, which are not "included" in another diagnosis.

Definition 3 (Minimal Diagnosis). A diagnosis D is minimal if no proper subset $D \quad D$ exists such that D is also a diagnosis.

The notions of diagnosis and minimal diagnosis are useful only in weak-fault models; these are models in which only the nominal behavior of the components is specified. If the opposite holds, then a superset of a minimal-diagnosis is not necessarily a diagnosis. We call *weak*, these models for which any superset of a diagnosis is also a diagnosis.

In models which specify faulty behavior (or, equivalently, unrestricted propositional theories) we define partial diagnosis to be any implicant of *SD*. Such partial diagnoses are more convenient in diagnostic reasoning (as they work on any theory), but their computation is more difficult [8].

Definition 4 (Partial Diagnosis). A term Δ over a set of assumable variables $h \quad COMPS$ is a partial diagnosis of $SD \quad OBS$ iff $\Delta \not\models SD \quad OBS$.

Similar to minimal diagnoses, we can impose a condition of minimality on partial diagnoses. Intuitively a kernel diagnosis is a partial diagnosis which does not contain a smaller diagnosis.

Definition 5 (Kernel Diagnosis). A partial diagnosis Δ is a kernel diagnosis iff no $\Delta \not\models SD \quad OBS$ exists, such that Δ is the conjunction of a proper subset of the literals in Δ.

The LYDIA language supports variables both in the Boolean and finite integer (FI) domains. The LYDIA toolkit proposes two approaches for unifying this – encoding FI variables as Booleans and vice-versa [12]. Encoding FI into Boolean is trivial and we will not discuss it. Working directly in the FI domain is a preferred option and below we introduce a multivalued representation very-close to the traditional Boolean one and suitable to conventional propositional algorithms (e.g., DPLL).

Definition 6 (Multi-Valued Literal). A multi-valued variable $v_i \quad V$ takes a value from a finite domain, which is an integer set $D_i = \{1,2,\ldots,m\}$. A positive multi-valued literal l_j^+ is a Boolean function $l_j^+ \quad (v_i = d_k)$, where $v_i \quad V, d_k \quad D_i$. A negative multi-valued literal l_j^- is a Boolean function $l_j^- \quad (v_i = d_k)$, where $v_i \quad V, d_k \quad D_i$. If not specified, a literal l_j can be either positive or negative.

It is possible to build multi-valued representation for each Boolean counterpart, resulting in multi-valued **Wff**, Multi-Valued Conjunctive Normal Form (MVCNF), Multi-valued Decision Diagrams (MDD), etc. Doing this often prevents combinatorial blow-ups caused by computationally expensive encodings.

Having these basic definitions of consistency-based diagnosis, we can compile our models to a number of representations. One should note that compilation does not decrease the

complexity of the diagnosis problem. Compilation can only trade between off-line time, on-line time and space. Our notion of off-line and on-line time is non-strict, depending on which representation is used at which phase of diagnosis. In the section that comes next we will discuss the properties of several very common representations, as well as some languages introduced for the first time in LYDIA and UPTIME.

Knowledge Compilation Map

As the complexity of diagnosis is well-known [24], there is no representation which supports diagnostic queries of interest (e.g. finding implicants and prime implicants) efficiently *both* in time and space. For this purpose we convert our models to a number of representations having different properties – some of them can check consistency in time linear of the size of the representation (but they require exponential time of the number of the variables in the theory for generating the representation), others have small size but require, in the worst-case exponential time for some queries of interest.

If, a model consists only of Boolean variables, it is straightforward to convert it to propositional theory. The implementation details of LYDIA, omitted in this paper for brevity, include variable and constraint rewriting, type-checking and system inlining. The result of this LYDIA model compilation process is a propositional theory.

Definition 7 (Propositional Wff). A propositional **Wff** is a formula over the literals $l_1, l_2, \ldots, l_n$, and the standard Boolean connectives $\neg, \ , \ , \ , \ .$

While it is possible to compute implicants (partial diagnoses) directly in propositional **Wff** [21], it is often convenient (requires simpler implementation) to convert the propositional model to a simpler representation like Conjunctive Normal Form.

Definition 8 (CNF). A propositional formula is in Conjunctive Normal Form (CNF) if it is a conjunction of disjunctions of literals, that is $= {}_1 \ {}_2 \ \ldots \ {}_n$, where ${}_i = (l_{i1} \ l_{i2} \ \ldots \ l_{im\ i})$ and l_{ij} is a negative or positive literal.

As we have seen in Section 2, models of engineered systems consist of the conjunction of multiple propositional formulae, each imposing constraints on the functioning of a component. This makes CNF an easy target for compiling LYDIA models. Unfortunately, finding all satisfiable assignments (the ultimate goal of an MBR reasoner) in a CNF (ALLSAT) is in **P#**. On the other extreme stays DNF, unfortunately converting CNF to DNF for underconstrained models results in a combinatorial blow-up.

Definition 9 (DNF). A propositional formula is in Disjunctive Normal Form (DNF) if it is a disjunction of conjunctions of literals, that is $= {}_1 \ {}_2 \ \ldots \ {}_n$, where ${}_i = (l_{i1} \ l_{i2} \ \ldots \ l_{im\ i})$ and l_{ij} is a negative or positive literal.

LYDIA and UPTIME use in fact more representation (as is visible from Figure 7), the CNF and DNF being the extremes as DNF can be exponentially bigger than an equivalent CNF but checking consistency in DNF is linear to the representation size which exponential in CNF.

The reason for the compilation map in Figure 7 is threefold. First, we need to reach logically equivalent representation which allows us to cross-validate the correctness of the LYDIA tools. The implementation of other state-of-the-art techniques (like CDA* [26]) allows us to verify the final diagnostic result and to compare the diagnostic performance under fair conditions. Finally, and most importantly, almost any translation causes combinatorial explosion with some models.

Models or submodels, depending on different characteristics (e.g., weak/strong modeling, etc.) can produce very different compilation results. More precisely, there are Boolean functions which have linear OBDD representation, but exponential irreducible CNF. Therefore it is beneficial to choose different representations for different models and parts of models. Full complexity analysis of all representations is impossible due to lack of space in this paper but is available in [5].

An important source of speed-up in modeling can be the model structure. The representations we have defined so far discard the hierarchical information which can lead to an explosion in the size of the compiled model. Therefore we introduce the hierarchical equivalents of our knowledge representation and in the next part of this section we will discuss an algorithm which utilizes this hierarchy for improving the efficiency of the diagnostic search.

Definition 10 (Hierarchical System). A hierarchical system is a rooted, edge-labeled, acyclic multidigraph $H = V, , E$, where every node V_i, $V_i \quad V$, contains a knowledge base SD_i and a set of components $COMPS_i$. The multidigraph is such that $COMPS_1 \ COMPS_2 \ \ldots \ COMPS_n =$. The root node is marked by and the labels of the edges in E are maps $f : SD_i \quad SD_j$ between the literals in the knowledge bases represented by the nodes V_i and V_j.

Some Diagnostic Algorithms

The first algorithm we show is similar to the CDA* in [26]. A multi-valued version of it, we use for all the IBDM experiments, shown in Section 4.

We assume that components failures are independent and use the *a priori* probability of a fault term to guide a heuristic search for the most likely diagnosis. The heuristic function for a candidate term is $f( ) = P_1(h_1)P_2(h_2)\ldots P_n(h_n)$, where $h_1, h_2, \ldots, h_n$ are all the health variables in . In this case the probability of h_i being true if h_i is in is $P(h_i)$ and $P_i(h_i) = 1 - P(h_i)$ if $\neg h_i$ is in . If neither h_i nor $\neg h_i$ are present in , $P_i(h_i) = max(P(h_i), 1 - P(h_i))$. Searching while using $f( )$ as a heuristics, allows us to construct the

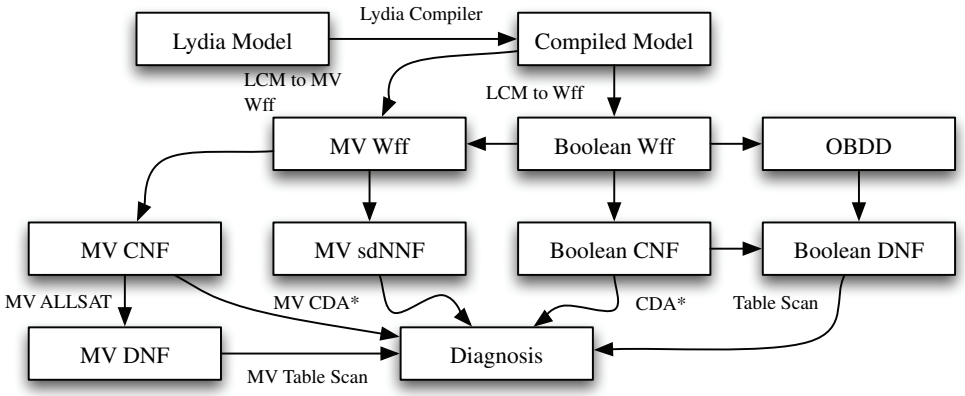

Figure 7. LYDIA and UPTIME translation map.

CDA* algorithm shown below.

Algorithm 1 CDA* in CNF.

```
1:  function CDA*(H)
        inputs: , a theory in CNF
        local variables: Q, priority queue
                         R, a set of terms
                         C, a set of conflicts
                         c, term
2:      PUSH(Q, INITIALSTATE( ))
3:      while (Δ  POP(Q)) =  do
4:          if Δ ≠  then
5:              R  R  Δ
6:              if TERMINATE() then
7:                  break
8:              end if
9:          else
10:             C  C  Δ
11:             REFINECONFLICTS(C)
12:         end if
13:         PUSH(Q, RESOLVINGSUCCESSORS(Δ, C))
14:     end while
15:     return R
16: end function
```

The auxiliary functions PUSH and POP perform the respective priority queue manipulation on Q (POP returns if the queue is empty). The initial state in the search tree, returned by INITIALSTATE, is the empty term. The RESOLVINGSUCCESSORS functions utilizes the heuristic function discussed above to return these candidate diagnoses which maximize the a priori probability and at the same time resolve all the conflicts accumulated in C.

For consistency checking (line 4) we can use DPLL [6] or, alternatively, an incomplete consistency checker like Binary-Constraint Propagation (BCP). It is interesting to note that even when we use the complete DPLL consistency checker, the consistency can be checked efficiently due to the fact that the problem is overconstrained.

The introduction of hierarchical systems allows us to similarly define hierarchical CNF and hierarchical DNF (the latter is not DNF anymore but is a restricted form of Negation Normal Form). We call this hierarchical DNF semi-decomposable Negation Normal Form (sdNNF). A hierarchical system is simply a conjunction of Boolean or multi-valued **Wff**. It is possible to discard this information (i.e., to *flatten out* the hierarchy) and to continue transforming the **Wff** in its flat representation.

Instead of flattening a hierarchical system we will *selectively apply compilations on subsystems* of *SD*. The need to exploit hierarchy stems from the inherent high-computational price of MBR. By exploiting the hierarchical information and selectively compiling parts of the model it is possible to increase the diagnostic performance and to trade cheaper pre-processing time for faster run-time reasoning. Our hierarchical algorithm, being sound and complete, allows large models to be diagnosed, where compile-time investment directly translates to run-time speedup.

Furthermore UPTIME *repartitions and coarsens* the original model in an attempt to minimize the subsystem connectivity, a process which leads to faster run-time fault diagnosis at the price of some pre-processing time. This is explained in the two algorithms below which show an advanced way for mixing compilation and hierarchical reasoning for fast diagnosis. The implementation of the two algorithms below is an important part of the UPTIME reasoning tool-kit.

Algorithm 2 shows the compilation of a LYDIA model to sdNNF. First it converts the model L to a hierarchical multi-valued NNF. Let the NNF N be a conjunction of **Wff** $N = {}_1 \quad {}_2 \quad \cdots \quad {}_n$. Then we build an interaction graph G in a fashion similar to [10] by having a node for each expression ${}_i$ in N and an edge in G if two expressions share a variable. We also weigh the edges of G with the number of variables two **Wff** ${}_i$ and ${}_j$ share.

Algorithm 2 Compilation of a LYDIA Model to sdNNF.

```
1:  function SDNNFFROMLYDIAMODEL(L,K,M)
        inputs: L, a LYDIA model
                K, number of partitions, integer
                M, maximum DNF size, integer
        local variables: G, interaction graph
                         P, a set of set of node indices
                         X, an OBDD
                         N = 1 2 ... n, an NNF
                         W, sdNNF, initially
2:      N  LYDIACOMPILE(L)
3:      G  INTERACTIONGRAPH(N)
4:      P  PARTITIONGRAPH(G,K)
5:      for all p P do
6:          X  NNFTOOBDD( i p i)
7:          W  W  OBDDTODNF(X))
8:      end for
9:      while , W :COUNTSOL( , ) < M do
10:         SDNNFNODESMERGE(W, , )
11:     end while
12:     return W
13: end function
```

Note that PARTITIONGRAPH is, itself, a computationally expensive process (it can be exponentially hard of the number of nodes in the graph G). LYDIA uses an approximation algorithm for graph partitioning [15] to solve the last problem. After the partitioning phase, we use an approximate model counter to merge nodes in the hierarchical description until any future merging would increase the number of terms in a node to a value exceeding a parameter M.

Next, we proceed with the run-time part, which is based on A*. We assume that components failures are independent and use the *a priori* probability of a fault term to guide a heuristic search for the most likely diagnosis. We assign the same small probability to all the components [7] as the reasoning technique is not probability-driven and it is possible to use other heuristics with similar results (e.g., the cardinality of a fault-mode).

Algorithm 3 computes diagnoses from the model produced by Algorithm 2 and *OBS*. The difference between the standard A* algorithm used in diagnosis, and this hierarchical version is that we try to find a conjunction of terms as opposed to conjunction of assumable variables consistent with *OBS*. The granularity of our approach is coarser and adjustable due to the parameterization of Algorithm 2 which allows trading space for time and in some cases reducing the overall computational complexity.

In this particular example Algorithm uses sdNNF but any hierarchical form with nodes consisting of tractable knowledge-bases will achieve similar results. In the main loop Algorithm 3 algorithm is chosen such a term c from the hierarchical node such that some heuristic estimate f(c) is maximized. When

Algorithm 3 A* search in sdNNF dictionary.

```
1:  procedure HIERARCHICALDIAGNOSE(H)
        inputs: H, hierarchical node
        local variables: Q, priority queue
                         s,c, terms
2:      PUSH(Q, INITIALSTATE(H))
3:      while (c POP(Q)) = do
4:          ENQUEUESIBLINGS(Q,c)
5:          if DIAGNOSIS(c) then
6:              OUTPUTDIAGNOSIS(c)
7:          else
8:              if (s NEXTBESTSTATE(H,c)) ≠ then
9:                  PUSH(Q,s)
10:             end if
11:         end if
12:     end while
13: end procedure
```

a consistent conjunction of terms is found from all the nodes in the hierarchy, OUTPUTDIAGNOSIS is invoked to send the result to the user.

4. EXPERIMENTS AND PERFORMANCE RESULTS

Test-Cases

This experimental section has two subsections. First, we discuss some high-level failure scenarios and the response of the diagnostic reasoner to them. Second, we analyze a number of component failures and check if they are correctly identified by the MBD loop.

High-Level Scenarios

Next we discuss some expected fault scenarios and how the UPTIME framework and the supplied IBDM model react to these scenarios.

Spurious "hard docking complete" signal by the IBDM— **Consequences:** The target spacecraft will resume attitude control and potentially break the link to the chaser spacecraft and may cause collision of both vehicles.

In this scenario we discuss a situation in which the Guidance, Navigation and Control (GNC) system supplies the diagnostic engine with a false signal indicating that the hard-docking system is in latched mode (the variable *dockingPhase* assumes the value *primaryLatched*). The result of the diagnosis is shown in Table 1.

Obviously, in this case the load measured by the load-sensing bolts of the latch tabs would be below the specified landmark value as specified in the observation list. The leading diagnostic result indicates that the root cause of the fault is a spurious signal which coincides with our diagnostic observation assumptions.

Symptoms (Observations)	
Variable	Value
dockingPhase	*primaryLatched*
latchTab[0].bolt.load	*below*
latchTab[1].bolt.load	*below*
⋮	
latchTab[11].bolt.load	*below*
Leading Diagnoses	
Variable	Value
controlState	*spuriousLatched*

Table 1. Diagnosis in case of a spurious "hard docking complete" signal by the IBDM.

Spurious "undocking complete" signal by the IBDM— **Consequences:** The chaser spacecraft will start its thrusters to initiate the departure while the physical link is still in place.

The scenario that follows is the opposite of the previous one. In this case the GNC system supplies the diagnostic engine with a false signal indicating that the hard-docking system is in unlatched mode (the variable *dockingPhase* has *primaryUnatched* or *secondaryUnlatched* value). The result of the diagnosis is shown in Table 2.

Symptoms (Observations)	
Variable	Value
dockingPhase	*primaryUnlatched*
latchTab[0].bolt.load	*above*
latchTab[1].bolt.load	*above*
⋮	
latchTab[11].bolt.load	*above*
Leading Diagnoses	
Variable	Value
controlState	*spuriousUnlatched*

Table 2. Diagnosis in case of a spurious "undocking complete" signal by the IBDM.

Again we have a diagnosis which clearly locates the root case of failure. The second diagnosis (not shown in this paper) is that all twelve latch tabs are faulty, which is less likely than to have a spurious "undocking complete" signal.

Inadvertent activation of the undocking sequence— **Consequences:** Potential depressurization of the target vehicle (habituated space elements only).

This test-case aims at diagnosing the unwanted situation in which the undocking sequence is activated outside of the planned release cycle. To allow the monitoring of such situations our model provides two observation variables: *dockingPhase* and *controlDockingPhase*. In a practical implementation the values of the two observables should be supplied independently, or, for the latter, even manually. Normally, the value of the *dockingPhase* variable is requested automatically from the GNC, and if it differs from the *desired* state for the hard-docking system there is an indication of an inadvertent action.

Symptoms (Observations)	
Variable	Value
dockingPhase	*primaryUnlatching*
controlDockingPhase	*primaryLatched*
Leading Diagnoses	
Variable	Value
controlState	*inadvertentAction*

Table 3. Diagnosis in case of an inadvertent undocking.

Table 3 simulates a scenario in which the desired and automatically supplied docking phase differ. The conclusion of the diagnostic reasoner is that there is an inadvertent action[3]. Note that the diagnostic solver we use for this paper (part of the UPTIME framework) is designed to reach such "single fault" conclusions for a very small amount of computational time. This would allow such an undesired scenario to be stopped and reversed in a timely fashion.

Loss of capability to open the hard docking link— **Consequences:** May cause severe degradation or even loss of the target's mission.

In this scenario the hard-docking system is put in "primary latching" mode, but it fails preventing the completion of the hard-docking process. This results in inability to open the hard docking link endangering the completion of the mission. Loss of capability for opening the hard docking link may be a result of the failure of any component or a set of components. We illustrate a non-nominal latching process by assuming a non-increasing load in one of the load sensing bolts and *simultaneously* an increased current measured in one of the EMA's electric motors.

In Table 4 the leading diagnosis consists of a double-fault (the complexity of finding all diagnoses is exponential to the number of components in a system). The so computed primary diagnosis is a plausible explanation of the observation. This example suggests that it is possible to detect situations preventing latching and that any diagnosis indicating non-nominal behavior, would result in the loss of capability of completing the hard-docking phase.

Component Failure Scenarios

Next, some typical scenarios will be discussed in which a component or a set of components malfunction. The main task of diagnosis is, then, to identify the root cause of fail-

[3] The task of inferring that this is a case of inadvertent undocking we leave to the human controller or to a future version of the model.

Symptoms (Observations)	
Variable	Value
dockingPhase	*primaryLatching*
latchTab[5].bolt.fderSign	*zero*
primaryEMA.current	*above*
⋮	⋮
Leading Diagnoses	
Variable	Value
primaryGearBoxState[8]	*jammed*
latchTab[5].state	*faulty*

Table 4. Non-nominal latching behavior illustrating a loss of capability for hard-docking and consequently opening the hard-docking link.

ure providing an explanation of the observed behavior. We illustrate the workings of our technique by choosing open- or short-circuited micro-switches, faulty latch tabs, jammed gear-boxes and latch assemblies, etc. In this paper we illustrate our approach with faults of small cardinality but the mechanism is equally applicable in cases when multiple components are malfunctioning simultaneously.

We start with a single short-circuited micro-switch scenario (cf. Table 5). It is enough to configure the docking system in "latched" state, to arbitrarily choose a latch assembly and to supply a "closed" value for one of the open primary mechanism micro-switches. As the latch mechanism is engaged the primary crank is in "latched" mode, hence the position of the micro-switches detecting the "unlatched" latch assemblies should be in "open" state. We have a discrepancy and most likely a faulty switch.

Symptoms (Observations)	
Variable	Value
dockingPhase	*primaryLatched*
structuralLatchAssembly[0]. primaryLimitSwitchAssemblyOpened. limitSwitch[0].reading	*open*
Leading Diagnoses	
Variable	Value
structuralLatchAssembly[0]. primaryLimitSwitchAssemblyOpened. limitSwitch[0].state	*shortCircuited*

Table 5. Diagnosis in case of a short-circuited micro-switch.

Actually, the diagnostic engine does not need the state of the mechanism in order to determine that there is a short-circuited micro-switch. Due to the high redundancy in the latch position sensors, there are four micro-switches for the "latched" and "unlatched" positions of the primary and secondary latch mechanisms. If one switch is faulty, the position of the other three will supply enough information for the reasoning engine to determine the root cause of failure.

The diagnostic process is less intuitive when we have two short-circuited micro-switches in the same latch assembly and limit-switch block. Such a scenario is intuitively less likely and it is also more computationally expensive to detect it (due to the a priori probability heuristics [26] used in our search algorithm). Table 6 illustrates such a case.

Symptoms (Observations)	
Variable	Value
dockingPhase	*primaryLatched*
structuralLatchAssembly[0]. primaryLimitSwitchAssemblyOpened. limitSwitch[0].reading	*open*
structuralLatchAssembly[0]. primaryLimitSwitchAssemblyOpened. limitSwitch[1].reading	*open*
Leading Diagnoses	
Variable	Value
structuralLatchAssembly[0]. primaryLimitSwitchAssemblyOpened. limitSwitch[0].state	*shortCircuited*
structuralLatchAssembly[0]. primaryLimitSwitchAssemblyOpened. limitSwitch[1].state	*shortCircuited*
primaryGearBoxState[0]	*jammed*

Table 6. Diagnoses in case of two short-circuited micro-switches.

In the case of the double-failure symptom (two short-circuited micro-switches in the same assembly) we have the most-likely diagnosis being, indeed, two faulty switches. The second diagnosis, however, is also plausible and that is a "jammed" gear-box (the third diagnosis is a "jammed" latch assembly but the hard-docking system lacks the ability of discerning faults in the gear-boxes from mechanical faults in the latch assemblies). Note, that in the "latched" and "unlatched" states of the system it is not possible to obtain a diagnostic information indicating a jammed "mechanisms" from the EMAs as the motors are switched-off (i.e., there is no measurement of the current supplied to their motors).

Following the examples with short-circuited limit-switches we briefly discuss two similar scenarios in which the micro-switches are open-circuited, that is, they do not indicate correctly a closed state when they are in a closed position.

Table 7 shows a case in which one of the switches of a primary latch assembly's switch block does not make contact in the primary latched position of the mechanism. The fault-diagnostic engine correctly identifies the cause of the error.

Table 8 demonstrates the failure of two switches in a single latch assembly's switch board. Similar to the case of the two short-circuited micro-switches (cf. Table 6) the first two diagnoses are of interest: two faulty switches and a jammed gear-box. The second diagnosis was not simulated in the original

Symptoms (Observations)	
Variable	Value
dockingPhase	*primaryLatched*
structuralLatchAssembly[0]. primaryLimitSwitchAssemblyClosed. limitSwitch[0].reading	*closed*
Leading Diagnoses	
Variable	Value
structuralLatchAssembly[0]. primaryLimitSwitchAssemblyClosed. limitSwitch[0].state	*openCircuited*

Table 7. Diagnoses in case of an open-circuited micro-switch.

Symptoms (Observations)	
Variable	Value
dockingPhase	*primaryLatched*
structuralLatchAssembly[0]. primaryLimitSwitchAssemblyClosed. limitSwitch[0].reading	*closed*
structuralLatchAssembly[0]. primaryLimitSwitchAssemblyClosed. limitSwitch[1].reading	*closed*
Leading Diagnoses	
Variable	Value
structuralLatchAssembly[0]. primaryLimitSwitchAssemblyClosed. limitSwitch[0].state	*openCircuited*
structuralLatchAssembly[0]. primaryLimitSwitchAssemblyClosed. limitSwitch[1].state	*openCircuited*
primaryGearBoxState[0]	*jammed*

Table 8. Diagnoses in case of two open-circuited micro-switches.

fault-scenario, but is a plausible explanation of the observed behavior.

In the next two cases we turn our look into the latch tabs. We will analyze cases when the hard-docking system is *in the process* of latching or unlatching but the load-sensing bolts (cf. Section 2) do not supply load information indicating nominal behavior. Once again, for brevity, we discuss only cases with single or double faults but the model-based framework is capable of correctly handling higher-cardinality faults.

A sound and complete search algorithm like the CDA* implemented in UPTIME provides *all* the explanations for an observed behavior. The order, however, depends on the *a priori* health variable probability assigned by the modeler. At this phase of the study the probability values are approximate and are used for ordering of the results only (and more importantly for guiding the search algorithm). This is contrasted to a fully probabilistic inference solver which can use the a posteriori probability of a fault, given an observation, for *quantifying* its belief for a *future* state of the system being analyzed.

Symptoms (Observations)	
Variable	Value
dockingPhase	*primaryUnlatching*
latchTab[0].bolt.fderSign	*minus*
latchTab[1].bolt.fderSign	*minus*
⋮	⋮
latchTab[10].bolt.fderSign	*minus*
latchTab[11].bolt.fderSign	*zero*
Leading Diagnoses	
Variable	Value
latchTab[11].state	*faulty*

Table 9. Diagnosis in case of a faulty latch tab.

In Table 9 the docking state is set to be in "primary unlatching" mode. In its nominal behavior we expect the load measured by the load-sensing bolts to be decreasing until a certain threshold is reached, and after this point, the docking-station is obviously in an "unlatched" régime. To simulate such an event it is enough to supply correct values for the load sensing time derivative (i.e., the load is decreasing in time) for all but one of the latch tabs.

The diagnostic engine determines the most probable fault to be a malfunctioning latch tab (or in this case a malfunctioning load-sensing bolt) – the same which has produced inconsistent observation for the state of unlatching.

Symptoms (Observations)	
Variable	Value
dockingPhase	*primaryUnlatching*
latchTab[0].bolt.fderSign	*minus*
latchTab[1].bolt.fderSign	*minus*
⋮	⋮
latchTab[9].bolt.fderSign	*minus*
latchTab[10].bolt.fderSign	*minus*
latchTab[11].bolt.fderSign	*minus*
Leading Diagnoses	
Variable	Value
latchTab[10].state	*faulty*
latchTab[11].state	*faulty*

Table 10. Diagnosis in case of two faulty latch tabs.

The scenario shown in Table 10 is similar to the previous one, except that it manifests a double-fault. In this case two load-sensing bolts measure non-decreasing load while unlatching. The diagnostic conclusion is that two faulty latch tabs are the most likely explanation of the observation. Note, that this information is based on the observations of the load-sensing bolts in the latch tabs of the *chaser vehicle only*. The diagnosis can be more soundly supported or refuted by information

from the target hard-docking system (i.e., faulty corresponding latch tab mechanisms in the chased vehicle would result in the same observation but healthy latch tabs in the chaser hard-docking mechanism).

The final scenario is a case in which we have a possibly obstructed movement in one or more of the hard-docking system mechanisms. We show the results of this experiment in Table 11.

Symptoms (Observations)	
Variable	Value
dockingPhase	*primaryLatching*
latchTab[0].bolt.fderSign	*plus*
latchTab[1].bolt.fderSign	*plus*
⋮	⋮
latchTab[11].bolt.fderSign	*plus*
primaryEMA.current	*above*
Leading Diagnoses	
Variable	Value
primaryGearBoxState[0]	*jammed*
primaryGearBoxState[1]	*jammed*
⋮	⋮
primaryGearBoxState[11]	*jammed*
primaryLatchState[0]	*jammed*
primaryLatchState[1]	*jammed*
⋮	⋮
primaryLatchState[11]	*jammed*
primaryEMA.stateMotor2	*faulty*

Table 11. Diagnoses in the event of a jammed gear-box or latch mechanism.

The hard-docking system is in the "latching" state for this experiment. The observation which conflicts with the nominal state of the system is the increased current consumption by a motor in the primary EMA. The IBDM hard-docking system states that an increased power consumption is an indication of a mechanical failure in one of the gear-boxes or in one of the latch-assemblies. The system lacks precision of locating the actual cause of failure as no observable parameter exists which would allow narrowing down the actual faulty mechanical mechanism. We show the first twenty-five diagnoses (note that according to the model based diagnosis theory an exponential number of diagnoses to the number of components may exist). These diagnoses are that either a faulty gear-box exists or a faulty latch-mechanism exists. Finally, it is also possible that the driving motor in the EMA is faulty.

Performance Evaluation

For the above experiments we have chosen to translate the LYDIA model to Multi-Valued Conjunctive Normal Form (MVCNF) [12] and then to perform A* style search in the compiled form. We note that LYDIA and UPTIME offer multiple chains of tools capable of computing the same diagnoses and the above selection of modules is not necessarily the optimal compilation path. Additional speedup can be achieved by exploiting compilation schemes like the ones cited in Section 1.

The total compilation time for our model, is below 1 s and is therefore negligible hence we omit compilation performance breakdown. The memory requirements are conservative as well, with the compiled MVCNF form containing 425 variables of which 161 health and 138 observables and a total of 1 328 clauses. A representation of the model, which has not been optimized for space occupies less than 100 Kb of RAM.

The results from the test-scenarios are shown in Table 12. These include all the examples from Section 4 and some additional cases. The search times vary from 100 ms to 250 s. The most computationally difficult cases require longer search due to the low probability of the faults, the high cardinality of the leading fault and the computation of many diagnoses.

All diagnoses with a leading single fault are computed in time less than 1 s, which indicates very good performance of the solvers in comparison to state-of-the art research [19], [26]. Unfortunately, at the time of writing of this paper, to our best knowledge no benchmark for comparing the performance of MBD solvers exist. Such a benchmark would allow a more precise comparison of the diagnosis performance to third-party implementations.

Scenario	N	Time [ms]
Spurious "Docking Complete" (cf. Table 1)	1	180
Spurious "Undocking Complete" (cf. Table 2)	1	233
Inadvertent Undocking (cf. Table 3)	1	166
Loss of Docking Capability (cf. Table 4)	1	37 112
SC Microswitch (cf. Table 5)	1	805
SC 2-Microswitches (cf. Table 6)	4	21 409
SC 3-Microswitches	2	21 486
SC 4-Microswitches	3	5 758
OC Microswitch (cf. Table 7)	1	242
OC 2-Microswitches (cf. Table 8)	2	5 651
OC 3-Microswitches	2	21 743
OC 4-Microswitches	2	21 078
Faulty Latch Tab (cf. Table 9)	1	253
Faulty 2 Latch Tabs (cf. Table 10)	2	246 663
Jammed Gearbox or Latch (cf. Table 11)	25	14 912

Table 12. Time for computing the first N diagnoses with different observation scenarios.

The performance data in this section is no substitute for a full performance analysis of LYDIA and UPTIME but an indication that the selection of tools offers computation of diagnosis in acceptable time.

All the experiments described in this paper are performed on a 3 GHz Pentium IV CPU.

5. CONCLUSIONS

Docking of spacecraft is a complex operation involving multiple mechanisms; an operation which can lead to faults with potentially catastrophic consequences endangering mission success and human-life (in manned missions). In the event of a failure, a timely recovery action would be necessary to prevent such catastrophic scenarios. A human controller on the ground or an astronaut in one of the vehicles can be greatly facilitated if an automatic root cause of failure is computed by a MBD framework. Manual troubleshooting can be difficult or impossible in the limited amount of time available for computing a recovery action.

Model-based reasoning can reduce the time and cost for creating troubleshooting procedures and applied automatically can provide a centralized or distributed [20] mechanism for determining causes of failure. Our experience shows that such an approach is sound and reasonably fast for complex systems like the hard-docking system, the ATV propulsion system [2] and our suggestions is to continue with the experimentation and application of this emerging technology to more, bigger and more complex spacecraft subsystems.

The purpose of this paper is twofold. First, we analyze the process of creating a health-model of an IBDM hard-docking system. The health model, created in the LYDIA language, is annotated and the modeling decisions are motivated. We suggest the use of the same technique for creating more precise docking models as well as models for diagnosing other ATV [2] subsystems.

The second purpose of this paper is to illustrate the use of a model-based diagnosis framework with the model of the IBDM hard-docking system. A number of experiments were performed and the results analytically confirmed, demonstrating the soundness of the software packages and techniques.

In the future, analyzing recorded experimental data from ground factory testing of the hard-docking system would allow modeling with higher accuracy. We suggest the recording and analysis of time-series of, e.g., the current of the EMA's motors, the load in the load-sensing bolts, etc. These time series would be analyzed, and modeled for obtaining additional diagnostic precision and more important *prognosis* of faults *before* they actually happen.

We have demonstrated that qualitative models, similar to the one described in this paper, can facilitate the trouble-shooting of critical systems. A promising technology is the hybridization of model-based qualitative diagnosis with quantitative methods for obtaining both high-precision diagnosis and being able to reason in case of uncertainty. A particular interesting function in this respect is the prediction of failures. In general, prognostics would require additional signal processing and knowledge of the physics of failure to identify the precursors of system failures. A prime candidate in this respect would be the analysis of the EMA currents, as physical degradation would likely to be reflected in the frequency components of the signal.

REFERENCES

[1] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki. *Diagnosis and Fault-Tolerant Control*. Springer Verlag, 2003.

[2] André Bos. Software architecture for integrated vehicle health management (IVHM) systems. Technical Report ESA-IVHM-TN-003, Science and Technology BV, November 2005.

[3] Karl Brace, Richard Rudell, and Randal Bryant. Efficient implementation of a BDD package. In *Proc. DAC'90*, pages 40–45, 1990.

[4] Adnan Darwiche. New advances in compiling CNF into DNNF. In *Proc. ECAI'04*, pages 328–332, 2004.

[5] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *JAIR*, 17:229–264, 2002.

[6] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *JACM*, 7(3):201–215, 1960.

[7] Johan de Kleer. Using crude probability estimates to guide diagnosis. *Artificial Intelligence*, 45(3):381–291, 1990.

[8] Johan de Kleer, Alan Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[9] Johan de Kleer and Brian Williams. Diagnosing multiple faults. *JAI*, 32(1):97–130, 1987.

[10] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., 2003.

[11] Wigbert Fehse. *Automated Rendezvous and Docking of Spacecraft*. Cambridge University Press, 2003.

[12] Alexander Feldman, Jurryt Pietersma, and Arjan van Gemund. A multi-valued SAT-based algorithm for faster model-based diagnosis. In *Proc. DX'06*, June 2006.

[13] H. Hoos. SAT-encodings, search space structure, and local search performance. In *Proc. IJCAI'99*, pages 296–303, 1999.

[14] Benjamin Kuipers. *Qualitative reasoning: modeling and simulation with incomplete knowledge*. MIT Press, Cambridge, MA, USA, 1994.

[15] Burkhard Monien and Stefan Schamberger. Graph partitioning with the party library: Helpful-sets in practice. *SBAC-PAD*, pages 1550–1533, 2004.

[16] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998.

[17] Delft University of Technology. LYDIA*: A Lan-*

guage for sYstem DIAgnosis – World Wide Web Site. `http://fdir.org/lydia/`.

[18] Ann Patterson-Hine, Gordon Aaseng, Gautam Biswas, Sriram Narasimhan, and Krishna Pattipati. A review of diagnostic techniques for ISHM applications. In *Proc. ISHEM'05*, November.

[19] Barney Pell, Douglas E. Bernard, Steve A. Chien, Erann Gat, Nicola Muscettola, P. Pandurang Nayak, Michael D. Wagner, and Brian C. Williams. An autonomous spacecraft agent prototype. In *In Proc. AGENTS'97*, pages 253–261. ACM Press, 1997.

[20] Gregory Provan. A model-based diagnosis framework for distributed systems. In *Proc. DX'02*, pages 16–25, 2002.

[21] A. Ramesh, G. Becker, and N. Murray. CNF and DNF considered harmful for computing prime implicants/implicates. *JAR*, 18(3):337–356, 1997.

[22] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[23] The MathWorks, Inc. *Matlab & Simulink – World Wide Web Site*. `http://mathworks.com/`.

[24] Farrokh Vatan. The complexity of the diagnosis problem. Technical Report NPO-30315, Jet Propulsion Laboratory, California Institute of Technology, 2002.

[25] Brian Williams and Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *Workshop on Logic-Based Artificial Intelligence*, College Park, Maryland, 1999.

[26] Brian Williams and Robert Ragno. Conflict-directed A and its role in model-based embedded systems. *Journal of Discrete Applied Mathematics*, 2004.

Alexander Feldman *is a doctoral research fellow at Delft University of technology, while also interested in applying his research to a number of real-world projects. He has written a number of articles on model-based diagnosis and automated reasoning. His interests include model-based reasoning, qualitative modeling, automated planning, hybrid methods for diagnosis and automated problem solving. Alexander Feldman obtained an M.Sc. degree (cum laude) in computer science from Delft University of Technology in 2004.*

Marco Caporicci *is an aeronautical engineer from the University of Rome, Italy. After having worked in the computer and the aerospace industry in Italy, he joined in 1988 the European Space Agency technical centre, ESTEC, in Noordwijk, The Netherlands. At ESTEC he occupied various positions in structural engineering and space propulsion, working on the ESA satellite and space transportation programmes. In 1995, he moved to the ESA Head Office in Paris as Technology Engineer for Future Launch Systems first and later on as Head of the Future Launcher and Technology Programmes. In 2001 he moved back to ESTEC to manage the European participation in the NASA X-38/CRV vehicle. Since 2002 he is Head of the Atmospheric Re-entry and Human Transportation Division in the Directorate of Human Spaceflight, with responsibility on technology developments and operational vehicle preparation for LEO and exploration transportation missions.*

Oscar Gracia *is a mechanical engineer. He has been working in the development of space mechanisms, structural health monitoring and atmospheric re-entry vehicles. He is also interested on the software implementation of control systems for mechanical systems. He received his M.Sc. degree in Industrial Engineering from University of Zaragoza (Spain).*

André Bos *has more than 15 years experience in health management system development for applications that include the Joint Strike Fighter (Boeing) and the (European) aerospace systems. In 1999, André Bos cofounded Science and Technology BV, an information technology company focusing on health management and embedded software technology for the aerospace and defense industry. Dr. Bos obtained a Ph.D. in model-based design from Delft University of Technology in 1998.*