Empirical Evaluation of Diagnostic Algorithm Performance Using a Generic Framework

Alexander Feldman[1], Tolga Kurtoglu[2], Sriram Narasimhan[3], Scott Poll[4], David Garcia[5], Johan de Kleer[6], Lukas Kuhn[6], Arjan van Gemund[1]

[1] *Delft University of Technology, Delft, 2628 CD, The Netherlands*
{*a.b.feldman,a.j.c.vangemund*}*@tudelft.nl*
[2] *Mission Critical Technologies @ NASA Ames Research Center, Moffett Field, CA, 94035, USA*
tolga.kurtoglu@nasa.gov
[3] *University of California, Santa Cruz @ NASA Ames Research Center, Moffett Field, CA, 94035, USA*
sriram.narasimhan-1@nasa.gov
[4] *NASA Ames Research Center, Moffett Field, CA, 94035, USA*
scott.poll@nasa.gov
[5] *Stinger Ghaffarian Technologies @ NASA Ames Research Center, Moffett Field, CA, 94035, USA*
david.garcia@nasa.gov
[6] *Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA*
{*lukas.kuhn,dekleer*}*@parc.com*

ABSTRACT

A variety of rule-based, model-based and data-driven techniques have been proposed for detection and isolation of faults in physical systems. However, there have been few efforts to comparatively analyze the performance of these approaches on the same system under identical conditions. One reason for this was the lack of a standard framework to perform this comparison. In this paper we introduce a framework, called DXF, that provides a common language to represent the system description, sensor data and the fault diagnosis results; a run-time architecture to execute the diagnosis algorithms under identical conditions and collect the diagnosis results; and an evaluation component that can compute performance metrics from the diagnosis results to compare the algorithms. We have used DXF to perform an empirical evaluation of 13 diagnostic algorithms on a hardware testbed (ADAPT) at NASA Ames Research Center and on a set of synthetic circuits typically used as benchmarks in the model-based diagnosis community. Based on these empirical data we analyze the performance of each algorithm and suggest directions for future development.

1 INTRODUCTION

Fault Diagnosis in physical systems involves the detection of anomalous system behavior and the identification of its cause. Some key steps in diagnostic inference are fault detection (is the output of the system incorrect?), fault isolation (what is broken in the system?), fault identification (what is the magnitude of the failure?), and fault recovery (how can the system continue to operate in the presence of the faults?). To develop diagnostic inference algorithms requires expert knowledge and prior know-how about the system, models describing the behavior of the system, and operational sensor data. This problem is challenging for a variety of reasons including:

- incorrect and/or insufficient knowledge about system behavior
- limited observability
- presence of many types of faults (such as system, supervisor, actuator, or sensor faults; additive and multiplicative faults; abrupt and incipient faults; persistent and intermittent faults; etc.)
- non-local and delayed effect of faults due to dynamic nature of the system.
- presence of other phenomena that influence/mask the symptoms of faults (unknown inputs acting on system, noise that affects the output of sensors, etc.)

Several communities have attempted to solve the diagnostic inference problem using various methods. Some approaches have been:

- Expert Systems - These approaches encode knowledge about system behavior into a form that can be used for inference. Some examples are rule-based systems (Russell & Norvig, 2003) and fault trees (Kavčič & Juričić, 1997).
- Model-Based Methods - These approaches use an explicit model of the system configuration and behavior to guide the diagnostic inference. Some examples are Fault Detection and Isolation (FDI) methods (Gertler, 1998),

This is an open-access article distributed under the terms of the Creative Commons Attribution 3.0 United States License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Submitted 2/2010; published 7/2010.

statistical methods (Basseville & Nikiforov, 1993), and "Artificial Intelligence (AI)" methods (Reiter, 1987).

- Data-Driven Methods - These approaches use the data from representative runs to learn parameters that can then be used for anomaly detection or diagnostic inference for future runs. Some examples are Inductive Monitoring System (IMS) (Iverson, 2004), and Neural Networks (Sorsa & Koivo, 1998).
- Stochastic Methods - These approaches treat diagnosis as a belief state estimation problem. Some examples are Bayesian Networks (Lerner, Parr, Koleer, & Biswas, 2000), and Particle Filters (de Freitas, 2002).

Despite the development of such a variety of notations, techniques, and algorithms, efforts to evaluate and compare diagnostic algorithms (DAs) have been minimal. One of the major deterrents is the lack of a common framework for evaluating and comparing diagnostic algorithms. The establishment of such a framework would accomplish the following objectives:

- Accelerate research in theories, principles, modeling and computational techniques for diagnosis of physical systems.
- Encourage the development of software platforms that promise more rapid, accessible, and effective maturation of diagnostic technologies.
- Provide a forum for algorithm developers to test and validate their technologies.
- Systematically evaluate diagnostic technologies by producing comparable performance assessments.

Such a framework requires the following:

- A standard representation format for the system description, sensor data, and diagnosis result.
- A software run-time architecture that can run specific scenarios from actual system, simulation, or other data sources such as files (individually or as a batch), execute DAs, send scenario data to the DA at appropriate time steps, and archive the diagnostic results from the DA.
- A set of metrics to be computed based on the comparison of the actual scenario and diagnosis results from the DA.

In this paper, we present a framework that attempts to address each of the above issues. The framework architecture employed for evaluating the performance of DAs is shown in Fig. 1 and is called DXF. Major elements are systems under diagnosis, DAs, scenario-based experiments, and metrics. System catalogs specify topology, components, and high-level mode behavior descriptions, including failure modes. DXF provides a program for quantitatively evaluating the DA output against known fault injections using predefined metrics.

The current version of DXF and this paper address a class of abrupt failures such as the ones often observed in electrical power systems. Other types of failures, for example intermittent or continuous ones, are left for future work.

1.1 Contributions

The contributions of this paper are as follows:

- It introduces a benchmarking framework to be used for systematic empirical evaluation of diagnostic algorithm performance. Moreover, it defines and describes the main elements of the framework so that the benchmarking results can be applied to any arbitrary physical or synthetic system by using the architecture described in the paper.
- It provides a comprehensive set of empirical evaluation results in order to validate the proposed framework and to facilitate the understanding and comparative analysis of different diagnostic technologies.

1.2 Organization of the Paper

The rest of this paper is organized as follows. Section 2 contains related work. Section 3 presents DXF in detail including the representation languages used, the run-time architecture developed for experimentation, and the diagnostic performance metrics defined. Section 4 describes how the benchmarking was performed including a description of the two systems used, the faults injected, the DAs tested, and the results. Section 5 presents major assumptions made and issues observed. Finally, Section 6 presents the conclusions.

2 RELATED WORK

The development of monitoring and diagnostic technologies is of great interest to many applications. As these algorithms become more readily available, the necessity for assessing the performance of alternative diagnostic tools becomes important. As a result, there is an increasing need for a framework to evaluate of competing diagnostic technologies.

To address this need, several researchers have attempted to demonstrate benchmarking capability (Orsagh, Roemer, Savage, & Lebold, 2002; Roemer, Dzakowic, Orsagh, Byington, & Vachtsevanos, 2005; Bartyś, Patton, Syfert, de las Heras, & Quevedo, 2006). Among these, Bartyś et al. (2006) presented a benchmarking study for actuator fault detection and identification (FDI). This study, developed by the DAMADICS Research Training Network, introduced a set of 18 performance indices used for benchmarking FDI algorithms on an industrial valve-actuator system. The indices measure the temporal performance of detection and isolation decisions, as well as true and false detection and isolation rates, sensitivity, and diagnostic accuracy. This benchmark study uses real process data, and demonstrates how the performance indices can be calculated for 19 actuator faults using a single fault assumption.

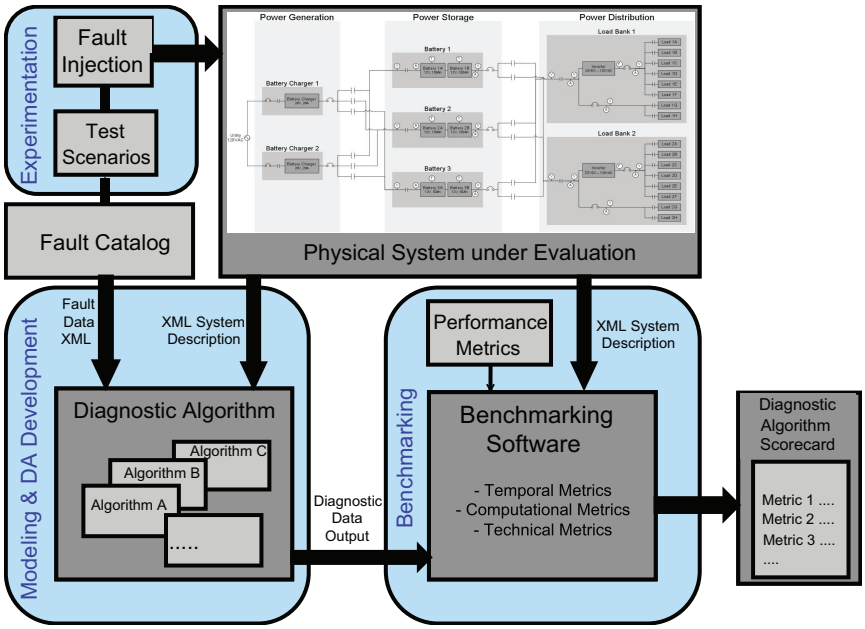

Figure 1: Framework architecture

Izadi-Zamanabadi and Blanke (1999) presented a ship propulsion system as a benchmark for autonomous fault control. This benchmark has two main elements. One is the development of an FDI algorithm, and the other is the analysis and implementation of autonomous fault accommodation.

Relevant to aerospace industry, Simon, Bird, Davison, Volponi, and Iverson (2008) introduced a benchmarking technique for gas path diagnosis methods to assess the performance of engine health management technologies.

Finally, Orsagh et al. (2002) provided a method to measure the performance and effectiveness of prognostics and health management algorithms for US Navy applications (Roemer et al., 2005). In this work, the performance metrics are defined separately for detection, isolation, and prognosis. In addition, this work also combined individual metrics into a composite score by implementing a weighted average sum. Moreover, it defined effectiveness metrics as a separate category that can be used to incorporate non-technical aspects such as operation, maintenance and implementation costs, computer resource requirements, and algorithm complexity into the analysis. Using these metrics, one can assess the overall effectiveness and benefit of diagnostic health management systems.

Other researchers have also proposed similar cost-benefit formulations for diagnostic systems (Williams, 2006; Kurien & Moreno, 2008; Hoyle, Mehr, Tumer, & Chen, 2007). These approaches, however, are primarily concerned with higher-level trade-offs in integrating diagnostic solutions to provide health management functionality and focus on performance indices such as operational cost, and maintainability.

The DXF framework presented in this paper adopts some of its metrics from Kurtoglu, Narasimhan, Poll, Garcia, Kuhn, de Kleer, van Gemund, and Feldman (2009) and extends prior work in this area by defining a number of novel diagnostic performance metrics; by providing a generic, application independent architecture that can be used for evaluating different monitoring and diagnostic algorithms; and by facilitating the use of real process data on a large-scale, complex engineering system.

3 FRAMEWORK

We have developed a framework called DXF that allows systematic comparison and evaluation of diagnostic algorithms under identical experimental conditions. The key components of this framework include representation languages for the physical system description, sensor data and diagnosis results, a runtime architecture for executing diagnostic algorithms and diagnostic scenarios, and an evaluation component that computes performance metrics based on the results from diagnostic algorithm execution.

The process to set up the framework in order to perform comparison/evaluation of a selected set of diagnostic algorithms on a specific physical system is as follows:

1. The system is specified in an XML file called the System Catalog. The catalog includes the system's components, connections, components' operating modes, and a textual description of component behavior in each mode.
2. The set of sensor points is chosen and sample data for nominal and fault scenarios are generated.

3. DA developers use the system catalog and sample data to create their algorithms using a predefined Application Programming Interface (API) in order to receive sensor data and send the diagnosis results. The DXF API is described later in this section.
4. A set of test scenarios (nominal and faulty) is selected to evaluate the DAs.
5. The run-time architecture is used to execute the DAs on the selected test scenarios in a controlled experiment setting, and the diagnosis results are archived.
6. Selected metrics are computed by comparing actual scenarios and diagnosis results from DAs. These metrics are then used to compute secondary metrics.

In the following subsections we describe the constituent pieces of our framework in more detail. The next subsection describes the various representation languages defined for the framework. We then describe the run-time architecture including the sequence of events and the messages exchanged among the various components and finally we describe a set of representative metrics that measure diagnostic performance.

3.1 DXF Data Structures

In what follows we describe the syntax and semantics of the relevant DXF data structures as well as some design rationale.

3.1.1 System Description

We realize that it is impossible to avoid bias towards certain diagnostic algorithms and methodologies when providing system descriptions. Despite attempts to create a general modeling language[1], there is no widely agreed way to represent models and systems. On the other hand, designing a diagnostic framework which is fully agnostic towards the system description is impossible as there would be no way to communicate components or system parts and to compute diagnostic metrics. As a compromise, we have chosen a minimalistic approach, providing formal descriptions of the system topology and component modes only.

The formal part of the DXF system description does not provide all information for building a model. The user may be provided with non-formalized external information, e.g., nominal and faulty functionality of components. This information may be provided in textual, programmatic or any other well-understood format. In the future we may try to extend our XML schema in yet another attempt of providing a complete modeling language beyond interconnection topology.

The XML system description is primarily intended to provide a common set of identifiers for components and their modes of operation within a given system. This is necessary to communicate sensor data and diagnoses. Additionally, basic structural information is provided in the form of component connections. Behavioral information is limited to a brief textual description of each component and its modes, leaving DA developers to deduce behavior from the system's sample data. This is done to avoid bias towards any diagnostic approach.

System Topology: DXF uses a graph-like representation to specify the physical connectivity of the system where nodes represent components of a system and arcs capture the connectivity between components.

Component Types: Each component in a system description refers to a component type. Note that in DXF, sensors do not imply special assumptions, i.e., sensors fail in the same way as "ordinary" components. A sensor, of course, should specify the data type it returns in order for DXF to send sensor readings to the DA under evaluation. A component type contains at least the following information:

- a name (identifier)
- an optional (textual) description
- a flag which specifies if this component type is a sensor
- a reference to a data structure describing the modes for the components of this type (both nominal and faulty)
- (sensors only) a data type of the sensor
- (sensors only) a range of the sensor

Component Mode Groups: Component operating modes are organized in mode groups. More than one component can refer to the same specific group. Each component type specifies a mode group. Each mode in a mode group contains:

- a name (identifier)
- an optional (textual) description
- a flag specifying if the mode is nominal or faulty

The details of the system description formats are provided in Appendix B.

3.1.2 API Data Types

In DXF, the run-time communication is performed using a messaging framework. Messages are exchanged as ASCII text over TCP/IP. API calls for parsing, sending, and receiving messages are provided with the framework, but developers may choose to send and receive messages directly through the underlying TCP/IP interface. This allows developers to use their programming language of choice, rather than being forced into the languages of the provided APIs.

Every message contains a millisecond timestamp indicating the time at which the message was sent. Though there are additional message types, the most important messages for the purpose of performance evaluation are the sensor data message, command message, and diagnosis message, described below (the details of the messaging formats are provided in Appendix C):

[1] For examples see Feldman, Provan, and van Gemund (2007) and the references therein.

Sensor/Command Data: Sensor data are defined broadly as a map of sensor IDs to sensor values (observations). Sensor values can be of any type; currently the framework allows for integer, real, Boolean, and string values. The type of each observation is indicated by the system's XML catalog.

Commandable components contain an additional entry in the system catalog specifying a command ID and command value type (analogous to sensor value type). The command message represents the issuance of a command to the system. In the ADAPT system, for example, the message (EY144CL, true) signifies that relay EY144 is being commanded to close. EY144CL is the command ID, and true is the command value (in this case, a Boolean value).

Candidates: The diagnostic algorithm's output (i.e., estimate of the physical status of the system) is standardized to facilitate the generation of common data sets and the calculation of the performance metrics. The diagnostic message contains:

- a timestamp value indicating when the diagnosis has been issued by the algorithm
- a list of diagnostic candidates (a candidate fault set may include a single candidate with a single or multiple faults; or multiple candidates each with a single or multiple faults)
- a detection flag (Boolean) as to whether the diagnosis system has detected a fault
- an isolation flag (Boolean) as to whether the diagnosis system has isolated a candidate or a set of candidates

In addition, each candidate in the candidate set has an associated weight. Candidate weights are normalized by DXF such that their sum for any given diagnosis is 1.

3.2 Run-Time Architecture

Figure 2 shows an overview of the DXF run-time architecture, its software components and data flows.

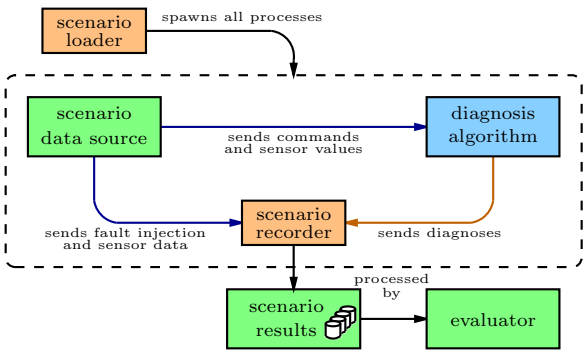

Figure 2: DXF run-time architecture

We next provide a brief description of each of the DXF's software components.

Scenario Loader (SL): SL is the main entry point for running the diagnostic scenarios. SL executes the Scenario Data Source, the Scenario Recorder, and all Diagnostic Algorithms. SL ensures system stability and clean-up upon scenario completion and is the only long-living process. The Scenario Data Source, Scenario Recorder and all Diagnostic Algorithms are spawned for each scenario and a Diagnostic Algorithm is forcibly killed if it does not terminate after a predetermined time-out.

Scenario Data Source (SDS): The SDS module provides scenario data from previously recorded datasets. The provenance of the data (whether hardware or simulation) depends on the system in question. A scenario dataset contains sensor readings, commands (note that the majority of classical MBD literature does not distinguish commands from observations), and fault injection information (to be sent exclusively to SR). SDS publishes data following a wall-clock schedule specified by timestamps in the scenario files.

Scenario Recorder (SR): SR receives fault injection data and diagnosis data into a results file. The results file contains a number of time-series which are described below. These time-series are used by the evaluation module for the computation of metrics. SR is the main timing authority, i.e., it timestamps each message upon arrival before recording it to the results file.

Diagnostic Algorithm (DA): A DA receives sensor and command data, performs diagnosis, and sends the diagnosis results back. As long as the DAs comply to the provided API, there are no restrictions on a DA; for example a DA may read precompiled data, or use external (user supplied) libraries, etc.

Evaluator: The evaluator computes a number of predefined metrics (see Sec. 3.3).

Consider the progression of a single diagnostic scenario. A typical one is shown in Fig. 3, where the fault injections, detection, and isolation are all treated as signals. These signals define a number of time points and intervals, as is seen below.

In the beginning of each scenario, a DA is given some startup time to initialize, read data, etc. Even though sensor observations could be available during startup, fault injections are not allowed during this interval. Fault injection and diagnosis take place during the diagnosis interval. Finally, a DA is given some shutdown time to terminate before being killed.

Table 1 summarizes the data collected by the SR for each scenario. These data are used for computing the various metrics discussed in Sec. 3.3. The time of first detection t_d is derived from the detec-

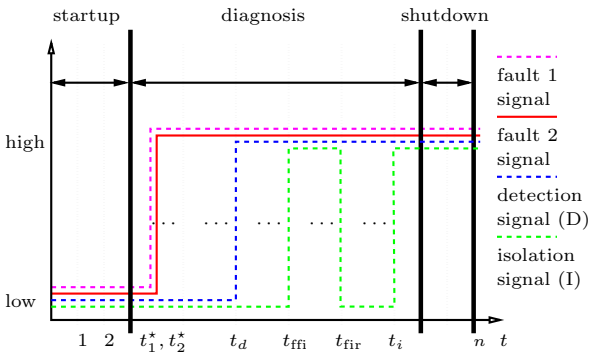

Figure 3: Signals and events during the development of a sample diagnostic scenario

tion signal D while the time of the last isolation is computed from the isolation signal I.

Var.	Type	Description	Origin
t_d	time-stamp	first detection	DA
t_i	time-stamp	last isolation	DA
C_s	real	startup CPU cycles	SL
C	time-series	CPU cycles per step	SL
M	time-series	memory in use	SL
$\omega^\star$	set	injected fault	SDS
$t^\star$	time-stamp	first fault injection	SDS
$t_i^\star$	time-stamp	injection of fault i	SDS
Ω	set of sets	candidate diagnoses	DA
W	set of reals	candidate weights	DA

Table 1: Scenario execution summary data

The set $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$ contains all diagnoses computed by the DA at time t_i. If a DA never asserts the isolation signal I (i.e., $t_i = \infty$), it is assumed that $\Omega = \emptyset$. Each candidate in Ω is accompanied by a weight W. We denote the set of weights of all diagnoses in Ω as $W = \{W(\omega_1), W(\omega_2), \ldots, W(\omega_n)\}$. The SR ensures that

$$\sum_{\omega \in \Omega} W(\omega) = 1 \tag{1}$$

by dividing each weight $W(\omega)$ with the sum of all weights. If a DA fails to provide W, it is assumed that all diagnoses are of the same weight.

In addition to the time-points defined in Table 1, the isolation signal in Fig. 3 shows the time t_{ffi} the DA has isolated a fault for the first time, and the time t_{fir} the DA has retracted its isolation assumption (for example because more faults are expected). Note that t_{ffi} and t_{fir} are not currently used by the evaluator for computing the metrics.

3.3 Diagnostic Performance Metrics

The metrics for evaluating diagnostic algorithm performance depend on the particular use of the diagnostic system, the users involved, and their objectives.

Several institutions and organizations have proposed metrics that measure diagnostic performance (Committee E-32, 2008; DePold, Siegel, & Hull, 2004; DePold, Rajamani, Morrison, & Pattipati, 2006; Metz, 1978; Orsagh et al., 2002; Roemer et al., 2005; Bartyś et al., 2006). Among those, the SAE's "Health and Usage Monitoring Metrics" (Committee E-32, 2008) defines probability of detection and probability of false alarms as key indices for evaluating diagnostic algorithm performance.

In Orsagh et al. (2002), the performance metrics are defined separately for detection, and isolation. For detection, the metrics include thresholds, accuracy, reliability, sensitivity to load, speed, or noise, and stability. The isolation metrics include the detection metrics, but also include measures for discrimination and repeatability.

In this paper, our goal has been to define a number of metrics and to give guidelines for their use. For DXF, we make a distinction between detection, isolation, and computational performance and highlight metrics for each category. In general several other classes of metrics are possible, including cost/utility metrics, effort (in building systems for example) metrics and also other categories such as fault identification and fault recovery metrics. The expectation is that as the DXF evolves a comprehensive list of desired metric classes and categories will be developed to aid framework users in choosing the performance criteria they want to measure.

Metric	Name	Class
M_{fd}	fault detection time	detection
M_{fn}	false negative scenario	detection
M_{fp}	false positive scenario	detection
M_{da}	scenario detection accuracy	detection
M_{fi}	fault isolation time	isolation
M_{err}	classification errors	isolation
M_{utl}	utility	isolation
M_{sat}	consistency	isolation
M_{cpu}	CPU load	computational
M_{mem}	memory load	computational

Table 2: Metrics summary

For the first implementation of the DXF, we defined 10 metrics which are summarized in Table 2. These metrics are based on extensive survey of literature and talking to experts from various fields (Kurtoglu, Mengshoel, & Poll, 2008). These metrics are defined next.

3.3.1 Detection Metrics

The distinction between detection and isolation has practical importance. A DA may announce a fault detection before it knows the root cause of failure (for example, a detection announcement can be based solely on surpassing sensor threshold values). A detection signal cannot be retracted by a DA while it is legal to retract an isolation an-

nouncement when more faults are expected. The detection metrics include:

Fault Detection Time The *fault detection time* (the reaction time for a diagnostic engine to detect an anomaly) is directly measured as:

$$M_{\text{fd}} = t_d \tag{2}$$

The fault detection time is reported in milliseconds and is computed only for non-nominal scenarios for which a DA asserts the time detection signal at least once.

False Negative Scenario The *false negative scenario* metric measures whether a fault is missed by a diagnostic algorithm and is defined as:

$$M_{\text{fn}} = \begin{cases} 1, & \text{if } t_d = \infty \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

False Positive Scenario The *false positive scenario* metric penalizes DAs which announce spurious faults and is defined as:

$$M_{\text{fp}} = \begin{cases} 1, & \text{if } t_d < t^\star \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where $t^\star = \infty$ for nominal scenarios (i.e., scenarios during which no fault is injected).

Note that the above two metrics (M_{fn} and M_{fp}) are computed for each scenario and their computation is based on the times of injecting and announcing the fault. We also have false negative and false positive components in the context of individual diagnostic candidates (recall that a DA sends a set of diagnostic candidates at isolation time) which we will discuss later in this paper.

Scenario Detection Accuracy The *scenario detection accuracy* metric is computed from M_{fn} and M_{fp}:

$$M_{\text{da}} = 1 - \max(M_{\text{fn}}, M_{\text{fp}}) \tag{5}$$

M_{da} is 1 if the scenario is true positive or true negative and 0 otherwise (equivalently, $M_{\text{da}} = 0$ if $M_{\text{fn}} = 1$ or $M_{\text{fp}} = 1$, and $M_{\text{da}} = 1$ otherwise). M_{da} splits all scenarios into "true" and "false". Incorrect scenarios are further classified into false positive (M_{fp}) and false negative (M_{fn}). Correct scenarios are true positive if there are injected faults and true negative otherwise (the latter separation into true positives and true negatives is rarely of practical importance).

3.3.2 Isolation Metrics

Computation of isolation metrics is more involved due to the fact that an isolation can be retracted. Furthermore, an isolation event contains a set of diagnostic candidates and we need metrics that compare this set of candidates to the injected fault. Accordingly, we have defined several metrics which are computed from the set of diagnostic candidates Ω and the injected fault ω^* (classification errors, and utility metrics). Consider a single diagnostic candidate $\omega \in \Omega$. Both the candidate ω and the injected fault $\omega^\star$ are sets of components. The intersection of those two sets are the properly diagnosed components. The false positives are the components that have been considered faulty but are not actually faulty. The false negatives are the components that have been considered healthy but are actually faulty. Figure 4 shows how ω and $\omega^\star$ partition all components into four sets.

False positives and false negatives in this context relate to individual candidates, i.e., misclassified components in a single diagnostic candidate. There are also scenario-based false negative and false positive metrics (defined earlier in this section), which summarize whole scenarios and are not to be confused with the false positives and false negatives in the context of isolation metrics.

For brevity we use the notation in Table 3 for the Fig. 4 sets.

Var.	Set	Description
f	$\lvert\text{COMPS}\rvert$	all components
n	$\lvert\omega^\star \setminus \omega\rvert$	false negatives
N	$\lvert\text{COMPS} \setminus \omega\rvert$	the set of healthy components from the viewpoint of the DA
$\bar{n}$	$\lvert\omega \setminus \omega^\star\rvert$	false positives
$\bar{N}$	$\lvert\omega\rvert$	the set of faulty components from the viewpoint of the DA

Table 3: Notation for sizes of some frequently used sets

Based on the representation given in Figure 4, the meaning of false positives and false negatives can be interpreted differently depending on what the diagnosis results are supporting (abort decisions, ground support, fault-adaptive control, etc.). Researchers have proposed different methods to assess the meaning of isolation accuracy and its practical and economical implications.

DePold et al. (2004) introduced metrics based on the receiving operating characteristic (ROC) analysis (Metz, 1978), which illustrates the tradeoff space between the probability of false alarm and the probability of detection for different signal to noise ratio (SNR) levels. The method is used to test the relative accuracy of diagnostic systems based on different threshold settings. Later, they also proposed a combined metric (DePold et al., 2006) that accounts for consequential event costs including missed detection, false alarms, and misdiagnosis. Another widely used metric for isolation accuracy is the Kappa Coefficient (Committee E-32, 2008). It is based on the construction of a confusion matrix that summarizes diagnostic results produced by a reasoner over a number of test/use cases. In essence, the Kappa Coefficient measures the ability of an algorithm to discriminate among many fault candidates.

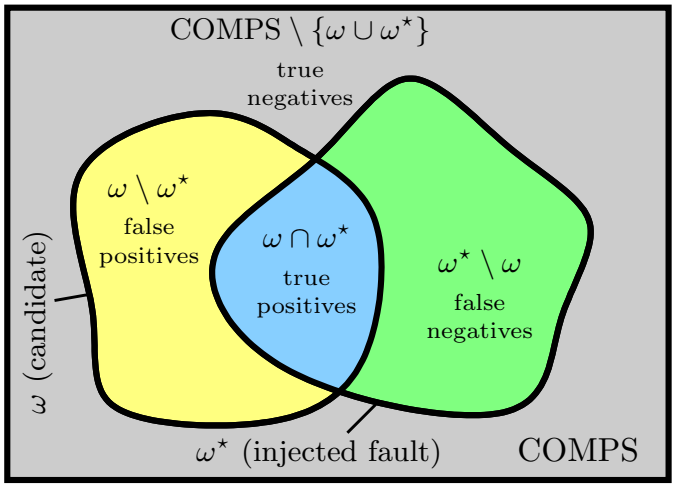

Figure 4: The diagnostic candidate ω and the injected fault $\omega^\star$ partition COMPS into four sets

In this paper, we take a simplistic approach and assume that false positives and false negatives have an equal cost for the diagnostic task and operations. The isolation metrics include (for a detailed discussion and derivation of the isolation metrics, see Appendix A):

Fault Isolation Time Consider an injected fault $\omega^\star = \{c_1, c_2, \ldots, c_n\}$ with the individual component faults injected at times $T^\star = \langle t_1^\star, t_2^\star, \ldots, t_n^\star \rangle$. Next, from the isolation signal, we construct a sequence of isolation times for each component. This sequence containing timestamps of rising edges of the isolation signal is denoted as T_i ($T_i = \langle t_1, t_2, \ldots, t_n \rangle$). Note that $t_k^\star < t_i$ for $1 \leq k \leq n$. The *fault isolation time* is then computed as:

$$M_{\text{fi}} = \frac{1}{n} \sum_{k=1}^{n} t_k - t_k^\star \tag{6}$$

If there is no isolation for specific fault (i.e., a fault is missed) then there is no difference $t_k - t_k^\star$ computed for that fault. E.g., if in a fault $\omega^\star = \langle c_1, c_2, c_3 \rangle$, c_1 is isolated, c_2 is not, and c_3 is; the isolation time $t_2 - t_2^\star$ is undefined and not included in the average ($n = 2$).

The *fault isolation time* is reported in milliseconds and is computed only for non-nominal scenarios for which a DA asserts the time isolation signal at least once.

Classification Error The *classification error* metric is defined as:

$$M_{\text{err}} = \sum_{\omega \in \Omega} W(\omega)(|\omega \ominus \omega^\star|) \tag{7}$$

In Eq. (7), $\omega \ominus \omega^\star$ denotes the symmetric difference of the ω and $\omega^\star$ sets, i.e., the number of misclassified components. Note that $|\omega \ominus \omega^\star| = n + \bar{n}$ and $f = N + \bar{N}$.

Utility The *utility* metric measures the work for correctly identifying all false negatives and false positives in a diagnostic candidate. Alternatively, the utility metric measures the expected number of calls to a testing oracle that always determines correctly the health state of a component. Note that this metric assumes an equal cost for fixing a false negative and a false positive. The derivation of the utility metric is given in Appendix A. The utility metric (per candidate) is:

$$m_{\text{utl}} = 1 - \frac{n(N+1)}{f(n+1)} - \frac{\bar{n}(\bar{N}+1)}{f(\bar{n}+1)} \tag{8}$$

Computing a weighted average of m_{utl} gives us the "per scenario" utility metric:

$$M_{\text{utl}} = \sum_{\omega \in \Omega} W(\omega) m_{\text{utl}}(\omega^\star, \omega) \tag{9}$$

The utility metric is, in fact, a combination of two "half-utilities"–the system repair utility and the diagnosis repair utility. The latter are defined as secondary metrics in Sec. 3.3.4 and discussed in detail in Appendix A.

Note that for $\Omega = \emptyset$, the framework automatically assumes a single "all-healthy" diagnostic candidate with weight 1 at the time of isolation. This affects the M_{err} and M_{utl} metrics. For example, in a non-nominal false-negative scenario, $M_{\text{err}} = |\omega^\star|$.

Consistency The next metric comes from MBD (de Kleer, Mackworth, & Reiter, 1992). It only applies to systems for which (1) there is a formally defined system description (model), (2) one can derive a formally defined observation from the sensor data, and (3) the notion of consistency is formally defined. We compute the consistency metric for the synthetic models and scenarios.

Consider a model SD and an observation α (α is derived from the sensor data at time t^*). If SD and α can be expressed as sentences in propositional

logic (as is the case with the synthetic models and scenarios) then the set of consistent diagnoses is defined as:

$$\Omega^{\top} = \{\omega \in \Omega : \text{SD} \wedge \alpha \wedge \omega \not\models \bot\} \tag{10}$$

The set $\Omega^{\top}$ can be computed from SD, α, and Ω by using a DPLL-solver (Davis, Logemann, & Loveland, 1962). The *consistency* metric can be computed from $\Omega^{\top}$, W and the injected fault $\omega^{\star}$:

$$M_{\text{sat}} = \sum_{\omega \in \Omega^{\top}} W(\omega) \tag{11}$$

M_{sat} is a measure of how much probability mass a DA associates with diagnoses consistent with the observations.

3.3.3 Computational Metrics

CPU Load The *CPU load* during an experiment is computed as:

$$M_{\text{cpu}} = C_s + \sum_{c \in C} c \tag{12}$$

where C_s is the amount of CPU time spent by a DA during startup and C is a vector with the actual CPU time spent by the DA at each time step. The CPU load is reported in milliseconds.

Memory Load The *memory load* is defined as:

$$M_{\text{mem}} = \max_{m \in M} m \tag{13}$$

where M is a vector with the maximum memory size allocated at each step of the diagnostic session. The memory load is reported in Kb.

3.3.4 Secondary Metrics

The intuition behind classification errors can be realized with multiple metrics. For example, a diagnostician may compute the isolation accuracy using:

$$M_{\text{ia}} = \sum_{\omega \in \Omega} W(\omega)(f - |\omega \ominus \omega^{\star}|) \tag{14}$$

In general a diagnostician has to perform extra work to "verify" all misdiagnosed components in ω. Suppose that the diagnostician has access to a test oracle that states if a component c is healthy or faulty. The system repair utility is then defined as normalized average number of oracle calls for identifying all false negative components and is defined as:

$$m_{\text{sru}} = 1 - \frac{n(N+1)}{f(n+1)} \tag{15}$$

The "per scenario" system repair utility is the defined as:

$$M_{\text{sru}} = \sum_{\omega \in \Omega} W(\omega) m_{\text{sru}}(\omega^{\star}, \omega) \tag{16}$$

Similarly, a diagnostician has to eliminate all false positive components in a candidate. This is reflected in the diagnosis repair utility:

$$m_{\text{dru}} = 1 - \frac{\bar{n}(\bar{N}+1)}{f(\bar{n}+1)} \tag{17}$$

The diagnosis repair utility for a set of diagnostic candidates is defined as:

$$M_{\text{dru}} = \sum_{\omega \in \Omega} W(\omega) m_{\text{dru}}(\omega^{\star}, \omega) \tag{18}$$

M_{utl}, M_{sru}, and M_{dru} are discussed in detail in Appendix A.

The choice of which utility metric is best for a particular use depends on the relative costs of the available repair actions. For example, if components are nearly free, but the act of replacing them is expensive then it makes no sense to identify which erroneously replaced components were actually correct (thus m_{sru} is preferred).

3.3.5 System Metrics

The metrics M_{fn}, M_{fp}, M_{da}, M_{fd}, M_{fi}, M_{err}, M_{utl}, M_{sat}, M_{cpu}, M_{mem}, M_{ia}, M_{sru}, and M_{dru} are based on a single scenario. To receive "per system" results we combine the metrics of each scenario using unweighted average. For example, if a system SD is tested with scenarios $\mathbf{S} = \{S_1, S_2, \ldots, S_n\}$, the "per system" utility of SD is computed as:

$$\bar{M}_{\text{utl}} = \sum_{S \in \mathbf{S}} \frac{1}{|\mathbf{S}|} M_{\text{utl}}(\text{SD}, S) \tag{19}$$

where $M_{\text{utl}}(\text{SD}, S)$ is the "per scenario" utility of system SD and scenario S.

The rest of the "per system" metrics ($\bar{M}_{\text{fn}}$, $\bar{M}_{\text{fp}}$, $\bar{M}_{\text{da}}$, $\bar{M}_{\text{fd}}$, $\bar{M}_{\text{fi}}$, $\bar{M}_{\text{err}}$, $\bar{M}_{\text{sat}}$, $\bar{M}_{\text{cpu}}$, $\bar{M}_{\text{mem}}$, $\bar{M}_{\text{ia}}$, $\bar{M}_{\text{sru}}$, and $\bar{M}_{\text{dru}}$) are defined in a way analogous to $\bar{M}_{\text{utl}}$.

Note that M_{fn}, M_{fp}, and M_{da} are called false negative scenario, false positive scenario and scenario detection accuracy, respectively. The analogous "per system" metrics $\bar{M}_{\text{fn}}$, $\bar{M}_{\text{fp}}$, and $\bar{M}_{\text{da}}$ are called *false negative rate*, *false positive rate*, and *detection accuracy*. $\bar{M}_{\text{da}}$, for example, represents the ratio of the number of correctly classified cases to the total number of cases. The latter "per system" metrics ($\bar{M}_{\text{fn}}$, $\bar{M}_{\text{fp}}$, and $\bar{M}_{\text{da}}$) are equivalent to the ones in Kurtoglu et al. (2009). In this paper we first define each metric "per scenario" and then "per system".

4 EMPIRICAL EVALUATION

In order to evaluate the framework presented in the previous section we selected two case studies. The first case study was performed on an Electrical Power System (EPS) testbed located in the ADAPT Lab of NASA Ames Research Center (Poll, Patterson-Hine, Camisa, Garcia, Hall, Lee, Mengshoel, Neukom, Nishikawa, Ossenfort, Sweet, Yentus, Roychoudhury, Daigle, Biswas, & Koutsoukos, 2007). This system mimics components and configurations in a power system that might be found on an aerospace vehicle. The second case study was performed on a set of 14 synthetic systems called the 74XXX/ISCAS85 circuits (Brglez & Fujiwara, 1985), which are purely combinational, i.e., they contain no flip-flops or

other memory elements, and represent well-known benchmark models of ISCAS85 circuits.

The empirical evaluation as part of the above two case studies employed 13 diagnostic algorithms (DAs) (Kurtoglu et al., 2009). The results from the DAs were used to compute metrics that were in turn used to evaluate the DAs performance on the aforementioned systems. We first present the DAs used in the evaluation and then present the two case studies.

4.1 Diagnostic Algorithms

We have experimented with a total of 13 DAs (see Table 4 for an overview). In what follows we provide a brief description of each DA.

DA	Systems	Algorithm Type
FACT	AL	model-based
Fault Buster	A,AL	statistical
GoalArt	A	flow-models
HyDE	A,AL	model-based
HyDE-S	AL	model-based
Lydia	S,A,AL	model-based
NGDE	S,AL	model-based
ProADAPT	A,AL	probabilistic
RacerX	AL	change detection
RODON	S,A,AL	model-based
RulesRule	AL	rule-based
StanfordDA	A	optimization
Wizards of Oz	A,AL	model-based

Table 4: Diagnostic Algorithms (S = synthetic, A = ADAPT, AL = ADAPT-Lite)

FACT: FACT (Roychoudhury, Biswas, & Koutsoukos, 2009) is a model-based diagnosis system that uses hybrid bond graphs, and models derived from them, at all levels of diagnosis, including fault detection, isolation, and identification. Faults are detected using an observer-based approach with statistical techniques for robust detection. Faults are isolated by matching qualitative deviations caused by fault transients to those predicted by the model. For systems with few operating configurations, fault isolation is implemented in a compiled form to improve performance.

Fault Buster: Fault Buster is based on a combination of multivariate statistical methods, for the generation of residuals. Once the detection has been done a neural network performs classification for computing isolation.

GoalArt: GoalArt Diagnostic System (Larsson, 1996) is based on multilevel flow models, which are crisp descriptions of flows of mass, energy, and information. It is a fast root cause analysis with linear computational complexity. Its main advantage is that it is very efficient to knowledge engineer a model. The algorithm has been proven in several commercial applications.

HyDE: HyDE (Hybrid Diagnosis Engine) (Narasimhan & Brownston, 2007) is a model-based diagnosis engine that uses consistency between model predictions and observations to generate conflicts which in turn drive the search for new fault candidates. HyDE uses discrete models of the system and a discretization of the sensor observations for diagnosis.

HyDE-S: HyDE-S uses the HyDE system but runs it on interval valued hybrid models and the raw sensor data.

Lydia: Lydia is a declarative modeling language specifically developed for Model-Based Diagnosis (MBD). The language core is propositional logic, enhanced with a number of syntactic extensions for ease of modeling. The accompanying toolset currently comprises a number of diagnostic engines and a simulator tool (Feldman, Provan, & van Gemund, 2009).

NGDE: An Allegro Common Lisp implementation of the classic GDE. NGDE (de Kleer, 2009) uses a minimum-cardinality candidate generator to construct diagnoses from conflicts. For ADAPT-Lite it uses interval constraints. No model of dynamics.

ProADAPT: ProADAPT (Mengshoel, 2007) processes all incoming environment data (observations from a system being diagnosed), and acts as a gateway to a probabilistic inference engine. The inference engine uses an Arithmetic Circuit evaluator which is compiled from Bayesian network models. The primary advantage of using arithmetic circuits is speed, which is key in resource bounded environments.

RacerX: RacerX is a detection-only algorithm which detects a percentage change in individual filtered sensor values to raise a fault detection flag.

RODON: RODON (Karin, Lunde, & Münker, 2006) is based on the principles of the General Diagnostic Engine (GDE) as described by de Kleer and Williams (1987) and the G$^+$DE (Heller & Struss, 2001). RODON uses contradictions (conflicts) between the simulated and the observed behavior to generate hypotheses about possible causes for the observed behavior. If the model contains failure modes in addition to the nominal behavior, these can be used to verify the hypotheses, which speeds up the diagnostic process and improves the results.

RulesRule: RulesRule is a rule-based isolation-only algorithm. The rule base was developed by analyzing the sample data and determining characteristic features of faults. There is no explicit fault detection though isolation implicitly means that a fault has been detected.

StanfordDA: StanfordDA is an optimization-based approach to estimating fault states in

DC power systems. The model includes faults changing the system topology along with sensor faults. The approach can be considered as a relaxation of the mixed estimation problem. The authors have developed a linear model of the circuit and pose a convex problem for estimating the faults and other hidden states. A sparse fault vector solution is computed by using L1 regularization (Zymnis, Boyd, & Gorinevsky, 2009).

Wizards of Oz: Wizards of Oz (Grastien & Kan-John, 2009) is a consistency-based algorithm. The model of the system completely defines the stable (static) output of the system in case of normal and faulty behavior. Given a new command or new observations, the algorithm waits for a stable state and computes the minimum diagnoses consistent with the observations and the previous diagnoses.

4.2 Case Study I: ADAPT EPS

We next describe the ADAPT EPS system, the diagnostic scenarios and the experimental results.

4.2.1 System Description

The ADAPT EPS testbed provides a means for evaluating DAs through the controlled insertion of faults in repeatable failure scenarios. The EPS testbed incorporates low-cost commercial off-the-shelf (COTS) components connected in a system topology that provides the functions typical of aerospace vehicle electrical power systems: energy conversion/generation (battery chargers), energy storage (three sets of lead-acid batteries), power distribution (two inverters, several relays, circuit breakers, and loads) and power management (command, control, and data acquisition).

The EPS delivers Alternating Current (AC) and Direct Current (DC) power to loads, which in an aerospace vehicle could include subsystems such as the avionics, propulsion, life support, environmental controls, and science payloads. A data acquisition and control system commands the testbed into different configurations and records data from sensors that measure system variables such as voltages, currents, temperatures, and switch positions. Data are presently acquired at a 2 Hz rate.

The scope of the ADAPT EPS testbed used in this case study is shown Fig. 5. Power storage and distribution elements from the batteries to the loads are within scope; there are no power generation elements defined in the system catalog. We have created two systems from the same physical testbed, ADAPT-Lite and ADAPT:

ADAPT-Lite ADAPT-Lite includes a single battery and a single load as indicated by the dashed lines in the schematic (Fig. 5). The initial configuration for ADAPT-Lite data has all relays and circuit breakers closed and no nominal mode changes are commanded during the scenarios. Hence, any noticeable changes in sensor values may be correctly attributed to faults injected into the scenarios. Furthermore, ADAPT-Lite is restricted to single faults.

ADAPT ADAPT includes all batteries and loads in the EPS. The initial configuration for ADAPT has all relays open and nominal mode changes are commanded during the scenarios. The commanded configuration changes result in adjustments to sensor values as well as transients which are nominal and not indicative of injected faults, in contrast to ADAPT-Lite. Finally, multiple faults may be injected in ADAPT. The differences between ADAPT-Lite and ADAPT are summarized in Table 5.

Aspect	ADAPT-Lite	ADAPT
|COMPS|	37	173
# of modes	93	430
relays initially	closed	open
initial state of the circuit-breakers	closed	closed
nominal mode changes	no	yes
multiple faults	no	yes

Table 5: ADAPT and ADAPT-Lite differences

4.2.2 Diagnostic Challenges

The ADAPT EPS testbed offers a number of challenges to DAs. It is a hybrid system with multiple modes of operation due to switching elements such as relays and circuit breakers. There are continuous dynamics within the operating modes and components from multiple physical domains, including electrical, mechanical, and hydraulic. It is possible to inject multiple faults into the system. Furthermore, timing considerations and transient behavior must be taken into account when designing DAs. For example, when power is input to the inverter there is a delay of a few seconds before power is available at the output. For some loads, there is a large current transient when the device is turned on. System voltages and currents depend on the loads attached, and noise in sensor data increases as more loads are activated. Measurement noise occasionally exhibits spikes and is non-Gaussian. The 2 Hz sample rate limits the types of features that may be extracted from measurements. Finally, there may be insufficient information and data to estimate parameters of dynamic models in certain modeling paradigms.

4.2.3 Fault Injection and Scenarios

ADAPT supports the repeatable injection of faults into the system in three ways:

Hardware-Induced Faults: These faults are physically injected at the testbed hardware. A simple example is tripping a circuit breaker using the manual throw bars. Another is using the power toggle switch to turn off an inverter. Faults may also be introduced in the loads attached to the EPS. For example, the valve can be closed slightly to vary the

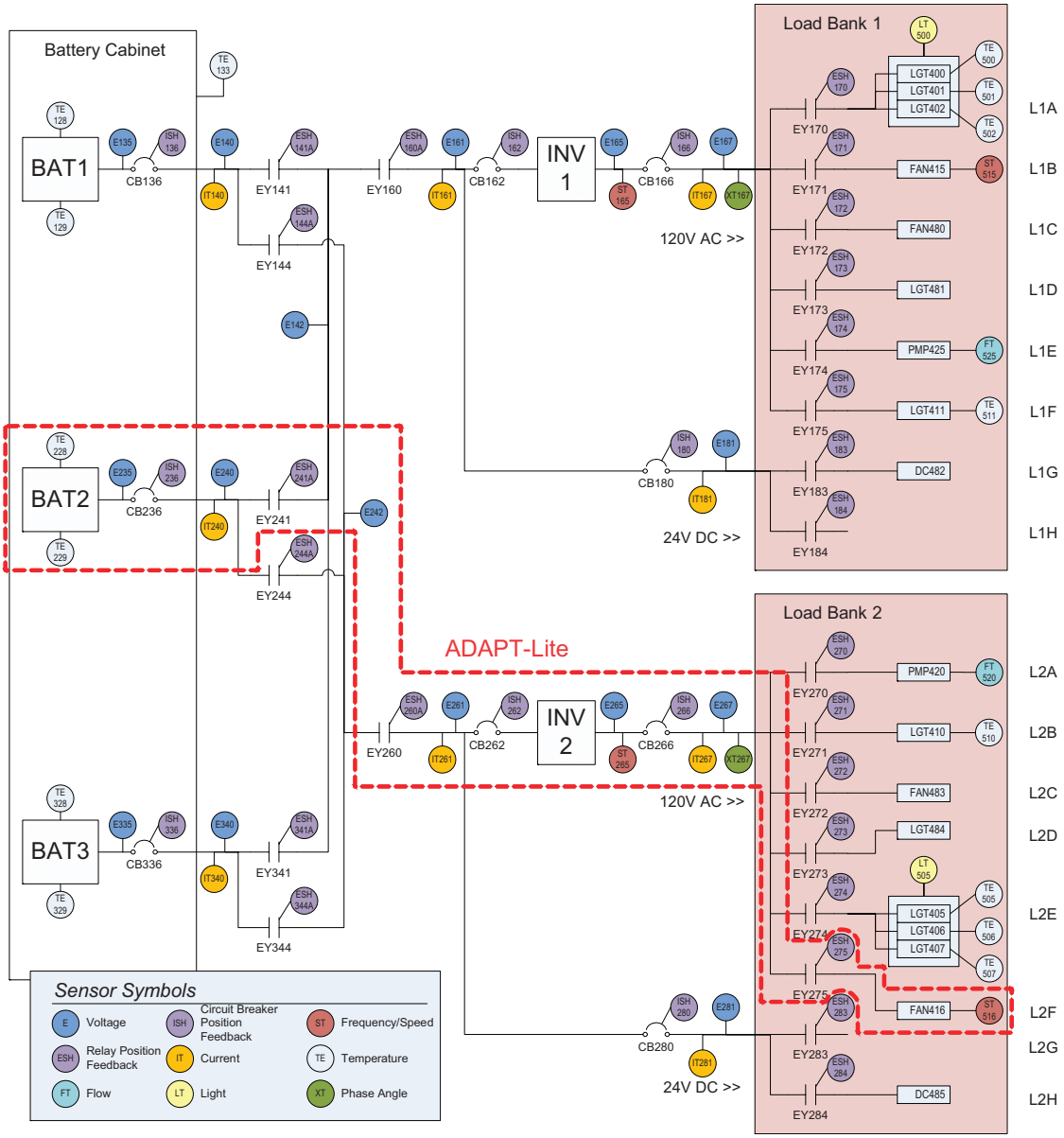

Figure 5: A schematic overview of the ADAPT EPS

back pressure on the pump and reduce the flow rate.

Software-Induced Faults: In addition to fault injection through hardware, faults may be introduced via software. Software fault injection includes one or more of the following: (1) sending commands to the testbed that are not intended for nominal operations; (2) blocking commands sent to the testbed; and (3) altering the testbed sensor data.

Real Faults: In addition the aforementioned two methods, real faults may be injected into the system by using actual faulty components. A simple example includes a burned out light bulb. This method of fault injection was not used in this study.

For results presented in this case study, only abrupt discrete (change in operating mode of component) and parametric (step change in parameter value) faults are considered. Nominal and failure scenarios are created using hardware and software-induced fault injection methods. The diagnostic algorithms are tested against a number of scenarios, each approximately four minutes in length.

The ADAPT-Lite experiments include 36 nominal and 56 single-fault scenarios. Table 6 summarizes the type of faults used for ADAPT-Lite.

Type	Subtype	Fault	#
battery	-	degraded	3
circ. breaker	-	failed-open	5
inverter	-	failed-off	2
load	fan	failed-off	2
		over-speed	2
		under-speed	2
relay	-	stuck-open	6
sensor	position	stuck	11
	current, phase angle, speed, temp., voltage	offset	12
		stuck	11
		Total:	56

Table 6: ADAPT-Lite faults

The ADAPT experiments have 48 nominal and 111 fault scenarios, which include single-fault, double-fault, and triple-faults. Figure 6 shows the fault-cardinality distribution of the ADAPT scenarios. Table 7 summarizes the type of faults used for ADAPT. The majority of faults involve sensors (102) and loads (30).

4.2.4 Experimental Results

We next compute the metrics described in Sec. 3.3 for the ADAPT-Lite and ADAPT scenarios.

ADAPT-Lite The DA benchmarking results for ADAPT-Lite are shown in Table 8, with graph-

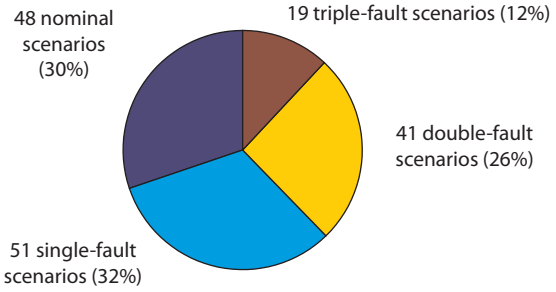

Figure 6: Fault-cardinality distribution of the ADAPT scenarios

Type	Subtype	Fault	#
battery	-	degraded	1
circuit breaker	-	failed-open	18
inverter	-	failed-off	10
load	basic	failed-off	1
	fan	failed-off	5
		over-speed	2
		under-speed	3
	light bulb	failed-off	14
	pump	failed-off	3
		blocked	2
relay		stuck-closed	3
		stuck-open	26
sensor	position	stuck	26
	current, flow, light, phase angle, speed, temp., voltage	offset	35
		stuck	41
		Total:	190

Table 7: ADAPT faults

ical depictions of some of the tabular data presented in Fig. 7. Figure 7 shows (1) M_{err} by DA (top-left), (2) M_{sru} and M_{dru} by DA (top-right), (3) M_{fd} and M_{fi} by DA (bottom-left), and (4) M_{fn} and M_{fp} (bottom-right). No DA dominates over all metrics used in benchmarking; nine of eleven DAs tested are best or second best with respect to at least one of the metrics.

The bottom-right plot of Fig. 7 shows the false positive and false negative rates. The corresponding detection accuracy can be seen in Table 8. As is evident from the definition of the metrics in Sec. 3.3, a DA that has low false positive and negative rates has high detection accuracy. False positives are counted in the following two situations: (1) for nominal scenarios where the DA declares a fault; and (2) for faulty scenarios where the DA declares a fault before any fault is injected. Noise in the data and incorrect models are the main causes

	Detection				Isolation			Computation	
DA	$\bar{M}_{\text{fd}}$	$\bar{M}_{\text{fn}}$	$\bar{M}_{\text{fp}}$	$\bar{M}_{\text{da}}$	$\bar{M}_{\text{fi}}$	$\bar{M}_{\text{err}}$	$\bar{M}_{\text{utl}}$	$\bar{M}_{\text{cpu}}$	$\bar{M}_{\text{mem}}$
FACT	1 785	0	0.11	0.89	10 798	11	0.975	15 815	4 271
Fault Buster	155	0.5	0.01	0.68	–	56	0.685	1 951	2 569
HyDE	13 355	0.46	0	0.72	13 841	45	0.79	23 418	5 511
HyDE-S	121	0.04	0.38	0.6	683	66	0.791	573	5 366
Lydia	232	0.18	0.01	0.88	232	100.3	0.785	1 410	1 861
NGDE	194	0.13	0.03	0.89	14 922	44.5	0.833	21 937	73 031
ProADAPT	4 732	0.05	0.01	0.96	7 104	10	0.955	1 905	1 226
RacerX	77	0.2	0.03	0.85	–	56	0.685	146	3 619
RODON	4 204	0.04	0.01	0.97	12 364	4	0.983	12 050	28 870
RulesRule	949	0.09	0.33	0.62	949	63	0.818	167	3 784
Wizards of Oz	12 202	0.5	0	0.7	12 327	43	0.769	1 153	1 682

Table 8: ADAPT-Lite metrics results

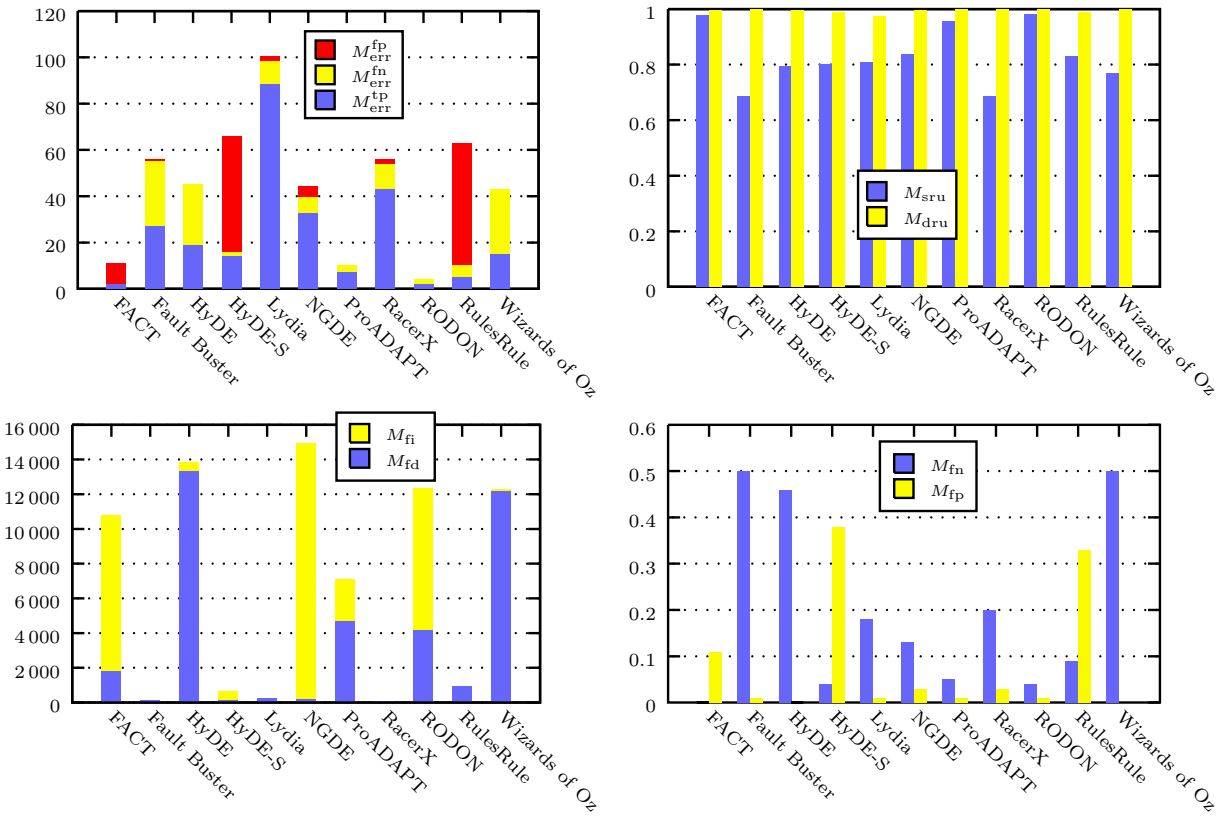

Figure 7: ADAPT-Lite metrics

of false positives. For example, the leftmost plot of Fig. 8 shows a nominal run with spike in sensor IT240 (battery 2 current); most of the DAs declare a false positive for this scenario. Many false negatives are caused by scenarios in which a sensor reading is stuck within the nominal range of the sensor. The middle plot of Fig. 8 shows an example of a sensor-stuck failure for voltage sensor E261, the downstream voltage of relay EY260.

The classification error metric for each DA is shown in the top-left plot of Fig. 7, where the error contributions of scenarios labeled false negative, false positive, and true positive are noted. Many DAs have difficulties distinguishing between sensor-stuck and sensor-offset faults. The distinction in the fault behavior is that stuck has zero noise while offset has the noise of the original signal; the rightmost plot in Fig. 8 shows the fan speed sensor ST516 with sensor-offset and sensor-stuck faults. In many scenarios, the sensor-stuck faults are set to the minimum or maximum value of the sensor or held at its last reading. The latter case presents the most difficulties to DAs.

M_{fd} and M_{fi} are shown in the bottom-left plot

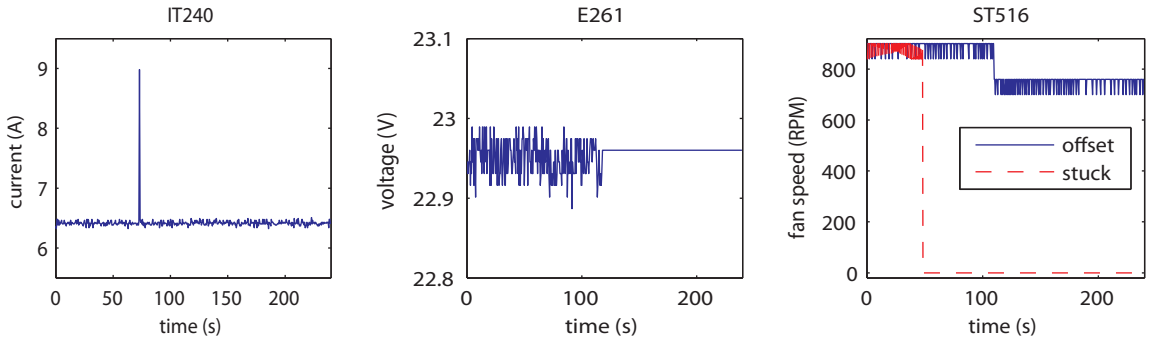

Figure 8: Examples of sensor readings

of Fig. 7. RacerX is a detection-only DA and does not perform isolation (its detection time is very low). Note that $M_{\text{fd}} \leq M_{\text{fi}}$, hence the bottom-left plot of Fig. 7 shows the isolation time stacked on the detection time (assume that part of the time first goes into detection and then into isolation).

The top-right plot of Fig. 7 shows the system repair utilty, M_{sru}, and the diagnosis repair utility, M_{dru}. The diagnosis repair utility is very close to 1 for all DAs, which reflects the small fault cardinality and diagnosis ambiguity groups for the system. The number of components that a DA considers faulty, $\bar{N}$, in any given scenario is typically close to the number of faults injected in the scenario. Since $\bar{N}$ is much less than the number of components, f, it is evident from equation (17) that M_{dru} approaches 1. Furthermore, since the number of healthy components, N, as determined by the DA is larger than the number of faulty components, $\bar{N}$, whereas n is typically not much different from $\bar{n}$, the system repair utility is smaller than the diagnosis repair utility.

Note that HyDE has been used by two different modelers of ADAPT-Lite. HyDE was modeled primarily with the larger and more complex ADAPT in mind and had a policy of waiting for transients to settle before requesting a diagnosis. The same policy was applied to ADAPT-Lite as well, even though transients in ADAPT-Lite corresponded strictly to fault events; this prevented false positives in ADAPT but negatively impacted the timing metric in ADAPT-Lite. On the other hand, HyDE-S was modeled only for ADAPT-Lite and did not include a lengthy time-out period for transients to settle. HyDE-S had dramatically smaller mean detection and isolation times (see the bottom-left plot of Fig. 7) with roughly the same M_{err} (see Table 8) as HyDE. This illustrates the impact that modeling and implementation decisions have on DA performance. While this gives some insight into trade-offs present in building models, in this work we did not define metrics that directly address the ease or difficulty of building models of sufficient fidelity for the diagnosis task at hand.

As is visible from Table 8, there exist significant differences in M_{cpu} and M_{mem}. Part of these differences can be attributed to the operating system (Linux or Windows$^{\text{TM}}$). RODON was the only Java DA that was run on Windows$^{\text{TM}}$, which adversely affected its memory usage metric.

ADAPT The empirical DA benchmarking results for ADAPT are shown in Table 9. Figure 9 shows (1) M_{err} by DA (top-left), (2) M_{sru} and M_{dru} by DA (top-right), (3) M_{fd} and M_{fi} by DA (bottom-left), and (4) M_{fn} and M_{fp} (bottom-right). Five of eight DAs tested were best or second best with respect to at least one of the metrics for ADAPT.

The comments in the ADAPT-Lite discussion about noise and sensor stuck apply here as well. Additionally, false positives also result from nominal commanded mode changes in which the relay feedback did not change status as of the next data sample after the command. Here is an extract from one of the input scenario files that illustrates this situation:

```
command @120950 EY275_CL = false;
sensors @121001 {..., ESH275 = true, ... };
sensors @121501 {..., ESH275 = false, ... };
```

A command is given at 120.95 seconds to open relay `EY275`. The associated relay position sensor does not indicate open as of the next sensor data update 51 milliseconds later. This is nominal behavior for the system. A DA that does not account for this delay will indicate a false positive in this case.

The detection and isolation times are generally within the same order of magnitude for the different DAs (see the bottom-left plot of Fig. 9). Some DAs have isolation times that are similar to its detection times while others show isolation times that are much greater than the detection times. This could reflect differences in reasoning strategies or differences in policies for when to declare an isolation based on accumulated evidence.

The CPU and memory usage are shown in Table 9. The same comment for RODON mentioned previously in regards to memory usage applies here. The convex optimization approach applied in the StanfordDA and the compiled arithmetic circuit in ProADAPT lead to very low CPU usages.

DA	Detection				Isolation			Computation	
	$\bar{M}_{\text{fd}}$	$\bar{M}_{\text{fn}}$	$\bar{M}_{\text{fp}}$	$\bar{M}_{\text{da}}$	$\bar{M}_{\text{fi}}$	$\bar{M}_{\text{err}}$	$\bar{M}_{\text{utl}}$	$\bar{M}_{\text{cpu}}$	$\bar{M}_{\text{mem}}$
Fault Buster	21 255	0.39	0.03	0.70	100 292	193	0.587	10 051	7 119
GoalArt	3 268	0.05	0.03	0.93	7 805	154	0.776	149	6 784
HyDE	15 612	0.31	0	0.79	20 114	174.3	0.668	28 807	19 135
Lydia	16 135	0.2	0.25	0.62	16 135	234.9	0.653	5 715	3 412
ProADAPT	1 743	0.02	0.09	0.90	23 544	57	0.915	4 260	778
RODON	5 543	0.03	0	0.98	35 792	75.6	0.853	85 331	31 459
StanfordDA	3 826	0.05	0.17	0.79	16 816	176.6	0.706	1 012	2 213
Wizards of Oz	25 695	0.09	0.16	0.77	50 980	209.2	0.76	17 111	3 390

Table 9: ADAPT metrics results

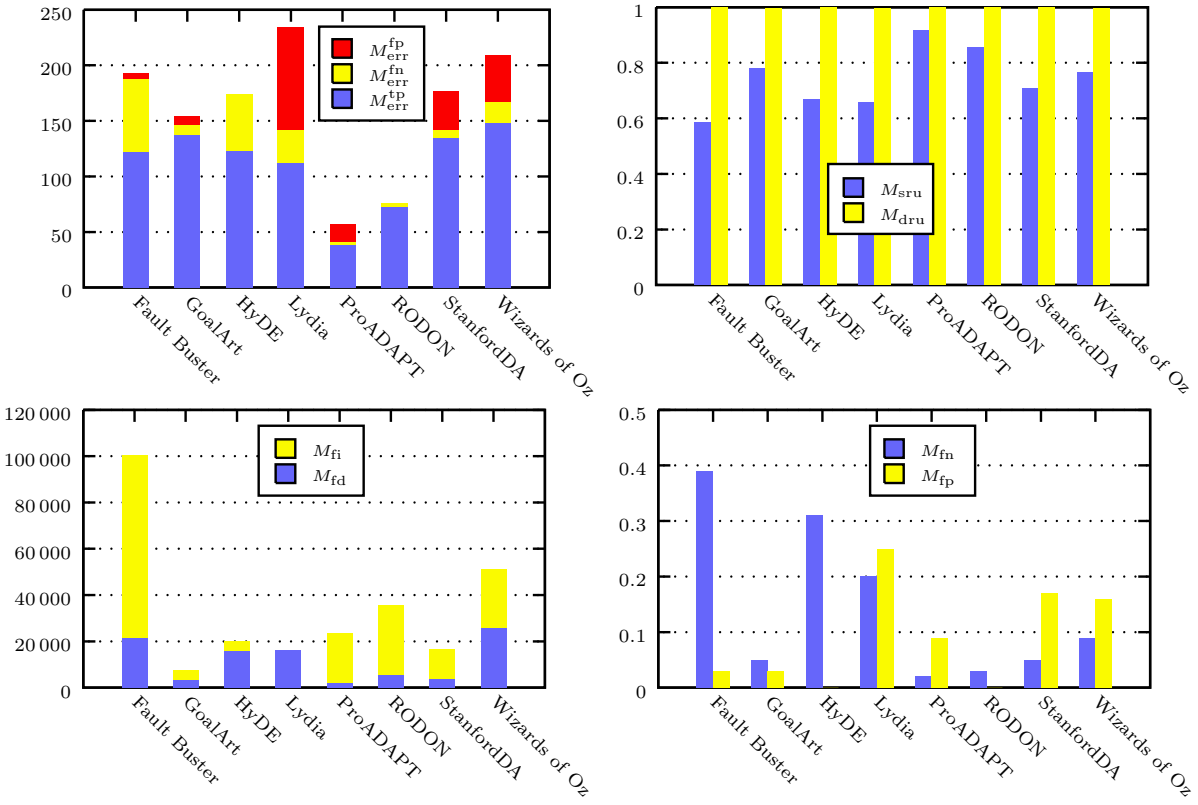

Figure 9: ADAPT metrics

4.2.5 Fault Type and Cardinality Analysis

The plots on the left-hand side of Fig. 10 show detection accuracy for all DAs by fault type for ADAPT-Lite and ADAPT. In general, M_{da} is not very sensitive to the component type, except in the case of load and sensor faults where it is lower. The data on the battery detection accuracy is not representative due to the limited number of fault scenarios containing battery faults (see Table 6 and Table 7).

The plots on the right-hand side of Fig. 10 show classification errors for all DAs by fault type for ADAPT-Lite and ADAPT. While the overall performance (averaged for all DAs) indicates that most fault categories result in roughly the same number of errors per scenario, it can be seen that a given DA may do better on some faults compared to others; furthermore, several DAs have the fewest classification errors for the different fault types. We should also note that in this benchmarking study, no partial credit was given for correctly naming the failed component but incorrectly isolating the failure mode. We realize however, that isolating to a failed component or line-replaceable-unit (LRU) in maintenance operations is sometimes all that is required. We plan to revisit this metric in future work.

Figure 11 shows the breakdown of classification errors by the number of faults in the scenario. In general, the number of errors increased approximately linearly with the number of faults in the

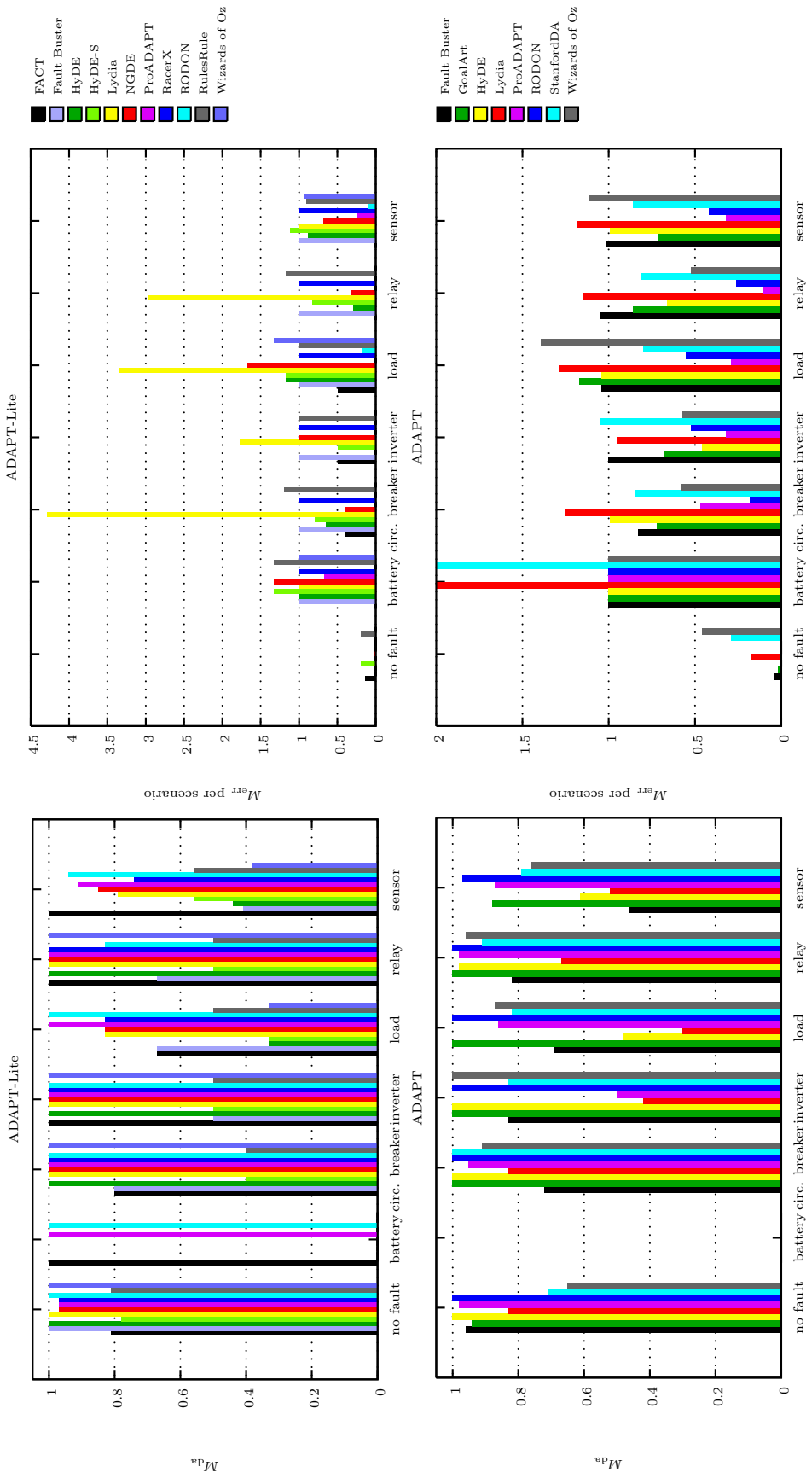

Figure 10: $\bar{M}_{\text{da}}$ and $\bar{M}_{\text{err}}$ per fault type for all DAs

scenario.The errors in the multiple fault scenarios were evenly divided among the faults; for example, if there were four classification errors in a scenario where two faults were injected, each fault was assigned two errors. We also did a more thorough assessment in which each diagnosis candidate was examined and classification erorrs were assigned to fault categories based on an understanding of which sensors are affected by the faults. The results are similar to evenly dividing the errors among the faults and are not shown here.

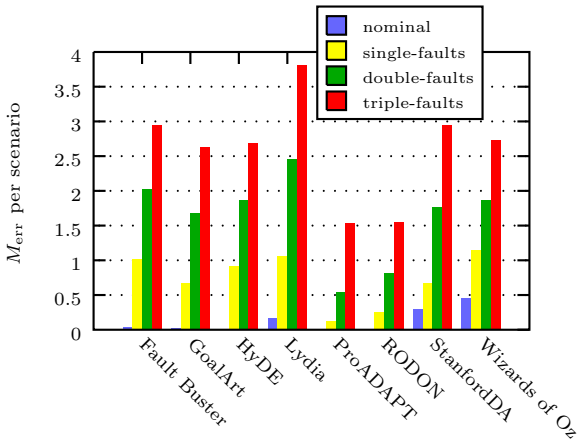

Figure 11: M_{err} per fault cardinality for all DAs (ADAPT)

4.2.6 Metric Correlations

The correlation matrix shown in Table 10 contains the Pearson's linear correlation coefficients between each metric for the industrial systems ADAPT and ADAPT-Lite.

Ideally, metrics should measure different aspects of DAs, i.e., the correlation matrix should contain small values only. Alternatively, users may use the correlation matrix from Table 10 to select metrics and adjust metric weights in computing the parameters of their DAs. Unexpected high correlations (or anti-correlations) between metrics indicate (1) bias due to the system or the sensor data, or (2) hidden metric dependencies.

All correlation coefficients in Table 10, except those shown in bold red, are significant–the p-values according to the Student's t distribution are smaller than 0.03.

Figure 12 is a color map of the correlation matrix from Table 10. Correlations or anti-correlations close to 1 are colored in red, while values closer to 0 are shown in blue colors.

The anti-correlation between M_{ia} and M_{err} is trivial (see (22)) and the only reason for including it is to show the correctness of our implementation.

The utility metric shows high correlation with the isolation accuracy/classification errors ($\rho = 0.75$). This is expected as both metrics measure similar properties of the DAs' results. Less trivial is the high correlation between M^{-}_{utl} and M^{+}_{utl} ($\rho = 0.84$). This indicates that DAs do not show

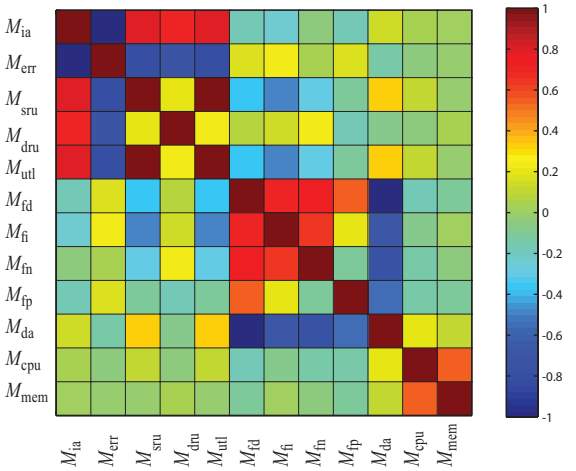

Figure 12: ADAPT & ADAPT-Lite metrics correlations

preferences towards diagnosing false negatives or false positives.

The time for fault isolation M_{ia} correlates highly with the three utility metrics, for which we have no explanation. The M_{fn} metric correlates high with M_{da} which comes from the metric design and indicates that, in general, DAs are tuned to avoid false positives at the price of more false negatives.

4.3 Case Study II: Synthetic Systems

We continue our discussion with an overview of the synthetic systems. The major differences between this case study and the previous are the sizes of the systems and the cardinalities of the injected faults. Furthermore, all system variables in this case study are of Boolean type. This case study aims to compare the robustness, CPU performance, and memory consumption of various DAs under stress conditions (large systems, faults of multiple-cardinality, etc.).

4.3.1 Description of Systems

The original 74XXX/ISCAS85 netlists can be mechanically translated into propositional **Wff**s. We have translated the propositional **Wff**s into logically equivalent Conjunctive Normal Form (CNF) formulae (Forbus & de Kleer, 1993). These CNF formulae are described in Table 11.

For each 74XXX/ISCAS85 CNF formula, Table 11 gives the number of inputs |IN|, the number of outputs |OUT|, the size of the components sets |COMPS|, the number of variables $|V|$, and the number of clauses $|C|$. The size of the 74XXX/ISCAS85 circuits can be reduced by using cones for computing single-component ambiguity groups (Siddiqi & Huang, 2007) or using fault collapsing.

4.3.2 Synthetic Model Scenarios

We have noticed that the performance of many DAs depends on the Minimum Cardinality (MC) of the diagnoses. Hence, we have performed our

	M_{ia}	M_{err}	M_{sru}	M_{dru}	M_{utl}	M_{fd}	M_{fi}	M_{fn}	M_{fp}	M_{da}	M_{cpu}	M_{mem}
M_{ia}	1	−1	0.79	0.7	0.8	−0.18	−0.25	**-0.04**	−0.18	0.15	**0.04**	**0.03**
M_{err}	−1	1	−0.79	−0.7	−0.8	0.18	0.25	**0.04**	0.18	−0.15	**-0.04**	**-0.03**
M_{sru}	0.79	−0.79	1	0.21	1	−0.37	−0.5	−0.31	−0.11	0.33	0.1	**-0.01**
M_{dru}	0.7	−0.7	0.21	1	0.24	0.07	0.14	0.23	−0.16	−0.09	**-0.04**	**0.06**
M_{utl}	0.8	−0.8	1	0.24	1	−0.36	−0.49	−0.3	−0.12	0.33	0.1	**0**
M_{fd}	−0.18	0.18	−0.37	0.07	−0.36	1	0.7	0.75	0.54	−0.98	−0.17	−0.12
M_{fi}	−0.25	0.25	−0.5	0.14	−0.49	0.7	1	0.65	0.19	−0.67	**-0.07**	**0.02**
M_{fn}	**-0.04**	**0.04**	−0.31	0.23	−0.3	0.75	0.65	1	−0.12	−0.76	−0.14	**-0.06**
M_{fp}	−0.18	0.18	−0.11	−0.16	−0.12	0.54	0.19	−0.12	1	−0.55	−0.13	−0.1
M_{da}	0.15	−0.15	0.33	−0.09	0.33	−0.98	−0.67	−0.76	−0.55	1	0.2	0.12
M_{cpu}	**0.04**	**-0.04**	0.1	−0.04	0.1	−0.17	**-0.07**	−0.14	−0.13	0.2	1	0.54
M_{mem}	**0.03**	**-0.03**	**-0.01**	**0.06**	**0**	−0.12	**0.02**	**-0.06**	−0.1	0.12	0.54	1

Table 10: ADAPT metrics correlation matrix (correlations with p-values smaller than 0.03 are shown in bold red)

Name	|IN|	|OUT|	|COMPS|	$\|V\|$	$\|C\|$
74182	9	5	19	47	75
74L85	11	3	33	77	118
74283	9	5	36	81	122
74181	14	8	65	144	228
c432	36	7	160	356	1 028
c499	41	32	202	445	1 428
c880	60	26	383	826	2 224
c1355	41	32	546	1 133	3 220
c1908	33	25	880	1 793	4 756
c2670	233	140	1 193	2 695	6 538
c3540	50	22	1 669	3 388	9 216
c5315	178	123	2 307	4 792	13 386
c6288	32	32	2 416	4 864	14 432
c7552	207	108	3 512	7 232	19 312

Table 11: 74XXX/ISCAS85 circuits

experimentation with a number of different observations leading to diagnoses of different MCs. Algorithm 1 generates observations leading to diagnoses of different MC, varying from 1 to nearly the maximum for the respective circuits (for the 74XXX models it is the maximum). The experiments omit nominal scenarios as they are trivial with synthetic systems.

The synthetic scenarios disregard the temporal aspects of diagnosis. They are created in the following way. In the beginning of a scenario, a DA is sent a nominal observation. After 5 s a fault $\omega^\star$ is injected. An observation α consistent with $\omega^\star$ is sent 6 s after the scenario start. We next discuss the generation of the "faulty" observations.

Algorithm 1 is an approximate algorithm that returns a set of observations A. Each observation $\alpha \in A$ leads to a diagnosis of different MC and is used in a different scenario. We have executed Alg. 1 multiple times, filtering out identical observations, until we have collected observations for a sufficient number of scenarios.

Algorithm 1 uses a number of auxiliary functions. RandomInputs (line 3) assigns uniformly

Algorithm 1 Algorithm for generation of observation vectors

1: **function** MakeAlphas(DS, N, K) **returns** a set of observations
   inputs: DS = ⟨SD, COMPS, OBS⟩
   OBS = IN ∪ OUT, IN ∩ OUT = ∅
   N, integer, number of tries
   K, integer, maximal number of diagnoses per cardinality
   local variables: $\alpha, \beta, \alpha_n, \omega$, terms
   c, integer, best card. so far
   Ω, set of terms, diagnoses
   A, set of terms, result
2: **for** $k \leftarrow 1, 2, \ldots, K$ **do**
3: $\alpha \leftarrow$ RandomInputs(IN)
4: $\beta \leftarrow$ NominalOutputs(DS, α)
5: $c \leftarrow 0$
6: **for all** $v \in$ OUT **do**
7: $\alpha_n \leftarrow \alpha \wedge$ Flip(β, v)
8: $\Omega \leftarrow$ Safari(SD, α_n, |COMPS|, N)
9: $\omega \leftarrow$ MinCardDiag(Ω)
10: **if** $|\omega| > c$ **then**
11: $c \leftarrow |\omega|$
12: $A \leftarrow A \cup \alpha_n$
13: **end if**
14: **end for**
15: **end for**
16: **return** A
17: **end function**

distributed random values to each input in IN (note that for the generation of observation vectors we partition the observable variables OBS into inputs IN and outputs OUT and use the input/output information which comes with the original 74XXX/ISCAS85 circuits for simulation). Given the "all healthy" health assignment and the diagnostic system, NominalOutputs (line 4) performs simulation by propagating the input assignment α. The result is an assignment β which contains values for each output variable in OUT.

The loop in lines 7 – 14 increases the cardinality by greedily flipping the values of the output variables. For each new candidate observation

α_n, Alg. 1 uses the diagnostic algorithm SAFARI to compute a minimal diagnosis of cardinality c (Feldman, Provan, & van Gemund, 2008a). As SAFARI returns more than one diagnosis (up to N), we use MINCARDDIAG to choose the one of smallest cardinality. If the cardinality c of this diagnosis increases in comparison to the previous iteration, the observation is added to the list.

By running Alg. 1 we get up to K observations leading to faults of cardinality $1, 2, \ldots, m$, where m is the cardinality of the MFMC diagnosis (Feldman, Provan, & van Gemund, 2008b) for the respective circuit. Alg. 1 clearly shows a bootstrapping problem. In order to create potentially "difficult" observations for a DA we require a DA to solve those "difficult" observations. In our case we have used the anytime SAFARI. As SAFARI is a stochastic algorithm, sometimes it returns a minimal diagnosis when we need a minimal-cardinality one. This leads to scenarios resulting in lower cardinalities than intended but this seemingly causes no problems except minor difficulties in the analysis of the DAs' performance.

4.3.3 Experimental Results

We start this section by computing the relevant metrics for this case study: $\bar{M}_{\text{utl}}$, $\bar{M}_{\text{cpu}}$, and $\bar{M}_{\text{mem}}$. The results are shown in Table 12.

It can be seen that LYDIA has achieved significantly better $\bar{M}_{\text{cpu}}$ and $\bar{M}_{\text{mem}}$ than NGDE and RODON. M_{utl} of LYDIA is slightly worse due to smaller number of diagnostic candidates computed by this DA. LYDIA and RODON showed similar results in the utility metrics.

We have computed $\bar{M}_{\text{sat}}$ and the results are shown in Table 14. The SAT and UNSAT columns show the number of consistent and inconsistent candidates, respectively. NGDE has generated approximately two orders of magnitude more satisfiable candidates than LYDIA and RODON. The policy of LYDIA has been to compute a small number of candidates, minimizing $\bar{M}_{\text{mem}}$ and $\bar{M}_{\text{cpu}}$. In order to improve $\bar{M}_{\text{utl}}$, LYDIA has mapped multiple-cardinality candidates into single-component failure probabilities. Hence, only single-fault scenarios contribute to the $\bar{M}_{\text{sat}}$ score for LYDIA.

5 DISCUSSION

The primary goal of the empirical evaluation presented in this paper was to demonstrate an end-to-end implementation of DXF and create a foundation for future usage of the framework. As a result we made several simplifying assumptions. We also ran into several issues during the course of this implementation that could not be addressed. In this section, we present those assumptions and issues, which we hope can be addressed in future implementations.

5.1 DXF Data Structures

The system catalog has been intentionally defined as a general XML format to avoid committing to specific modeling or knowledge representations (e.g., equations). It is expected that the sample training data and pointers to additional documentation would be sufficient for DA developers to learn the behavior of the system. We will continue to look for ways to extend the system catalog representation to provide as much general information about the system as possible. The diagnosis result format is defined to be a set of candidates with a weight associated with each candidate. Each candidate reports faulty modes of 0 (all nominal) or more components. Obviously this is a simplistic representation since it does not allow reporting of intermittent faults, parametric faults, among others. Also, in some cases it may be desirable to report a belief state (a probability distribution over component states) as opposed to a set of candidates.

5.2 Run-Time Architecture

For the ADAPT system, the fault signatures were limited to abrupt parametric and discrete types. We plan to introduce other fault types (incipient, intermittent, and noise) in the future. The runtime architecture was defined such that no assumptions were made regarding the actual operational environments in which the diagnostic algorithms may be run. We understand that a true test would simulate operating conditions of the real system, i.e. the system operates nominally for long periods of time and failures occur periodically following the prior probability of failure distribution. In this work, faults were inserted assuming equal probabilities. In the future, we will provide the failure rates of components and use these to evaluate the performance of DAs. It was also assumed that all sensor data was available to the DAs at all time steps. In the future, we would like to relax this assumption and provide only a subset of the sensor data. Additional ideas for future research include giving DAs reduced sensor sets, introducing multi-rate sensor data, injecting transient faults, allowing for autonomous transitions, adding variable loads, and extending the scope and complexity of the physical system.

For the synthetic systems, all the systems have been known in advance. This means researchers could optimize for these circuits. In addition, only one observation time was sampled. In the future, we will provide multiple observations. This will evaluate a DA's ability to merge information from multiple times. An important component of troubleshooting is introducing probe points. In the future, we can evaluate the number of probes needed to isolate the fault.

5.3 Diagnostic Metrics

The set of metrics we have chosen as primary is based on literature survey and expert opinion on what measures are important to assess the effectiveness of DAs. However, we realize that this set is by no means exhaustive. Different sets of metrics may be applicable depending on what the diagnosis results are supporting (abort decisions, ground support, fault-adaptive control, etc.). In addition there might be a set of weights associated with the metrics depending on their impor-

	Lydia			NGDE			RODON		
Name	$\bar{M}_{\text{utl}}$	$\bar{M}_{\text{cpu}}$	$\bar{M}_{\text{mem}}$	$\bar{M}_{\text{utl}}$	$\bar{M}_{\text{cpu}}$	$\bar{M}_{\text{mem}}$	$\bar{M}_{\text{utl}}$	$\bar{M}_{\text{cpu}}$	$\bar{M}_{\text{mem}}$
74182	0.365	62	17	0.466	230	10 716	0.262	1 293	18 205
74L85	0.455	53	18	0.575	341	11 838	0.372	5 233	22 533
74283	0.419	57	17	0.479	206	10 654	0.353	4 863	20 714
74181	0.374	73	21	0.486	213	10 879	0.405	14 222	26 962
c432	0.529	91	24	0.664	319	12 058	0.492	19 129	36 772
c499	0.29	80	33	0.414	1 719	17 063	0.258	20 649	36 436
c880	0.262	1 842	37	0.296	1 516	21 437	0.275	18 404	34 843
c1355	0.335	387	34	0.37	4 734	23 967	0.373	22 133	33 653
c1908	0.208	745	29	0.232	8 994	33 995	0.19	24 361	36 102
c2670	0.603	327	119	0.921	571	14 828	0.886	17 178	34 069
c3540	0.355	833	33	0.374	9 223	31 954	0.307	49 397	48 162
c5315	0.243	811	94	0.531	6 477	22 406	0.238	87 720	50 526
c6288	0.316	2 162	32	0.32	11 784	65 086	0.316	89 130	51 268
c7552	0.3	2 001	97	0.436	8 638	39 592	0.364	172 558	65 846
Averaged	0.361	680	43	0.469	3 926	23 320	0.364	39 019	36 864

Table 12: Synthetic systems metrics results

	Lydia			NGDE			RODON		
	$\bar{M}_{\text{sru}}$	$\bar{M}_{\text{dru}}$	$\bar{M}_{\text{err}}$	$\bar{M}_{\text{sru}}$	$\bar{M}_{\text{dru}}$	$\bar{M}_{\text{err}}$	$\bar{M}_{\text{sru}}$	$\bar{M}_{\text{dru}}$	$\bar{M}_{\text{err}}$
74182	0.381	0.984	69	0.574	0.892	78	0.262	1	80
74L85	0.458	0.996	30	0.617	0.958	39	0.46	0.913	78
74283	0.437	0.982	46	0.523	0.957	51	0.423	0.93	82
74181	0.378	0.995	48	0.517	0.969	55	0.456	0.949	87
c432	0.53	0.999	29	0.671	0.993	35	0.505	0.987	64
c499	0.293	0.997	71	0.428	0.986	78	0.268	0.99	107
c880	0.263	0.999	89	0.306	0.99	127	0.281	0.994	113
c1355	0.336	0.999	73	0.375	0.995	94	0.375	0.999	71
c1908	0.208	0.999	69	0.239	0.993	113	0.191	1	70
c2670	0.603	1	24	0.921	1	6	0.886	1	10
c3540	0.355	1	58	0.376	0.999	88	0.308	0.999	82
c5315	0.243	1	73	0.532	0.999	58	0.239	0.999	114
c6288	0.317	1	16	0.32	1	15	0.317	0.999	18
c7552	0.3	1	60	0.437	0.999	69	0.364	0.999	70
Averaged	0.364	0.996	54.02	0.488	0.981	64.75	0.381	0.983	74.71

Table 13: Synthetic systems secondary metrics results

tance (for abort decisions the fault detection time is of utmost importance). We expect to add more metrics to the list in the future (with support tools to compute those metrics). In addition since we were dealing with abrupt, persistent, and discrete faults, metrics associated with incipient, intermittent, and/or continuous faults were not considered.

Finally, the metrics listed in this paper do not capture the amount of effort necessary to build models of sufficient fidelity for the diagnosis task at hand. Furthermore, we have not investigated the ease or difficulty of updating models with new or changed system information. The art of building models is an important practical consideration which is not addressed in the current work.

In future work, we would like to determine a set of application-specific use cases (maintenance, autonomous operation, abort decision etc.) that the DA is supporting and select metrics that would be relevant to that use case.

5.4 Empirical Evaluation

Some practical issues arose in the execution of experiments. Much effort was put into ensuring stable, uniform conditions on the host machines; however, during the implementation, it was necessary to take measures that may have caused slight variability. One example was the manual examination of ongoing experiment results for quality assurance. Future releases of the DXF can address this by being more robust to unexpected DA behavior, and sending notifications in the event of such. Additionally, for Java DAs, significant dif-

	Lydia			NGDE			RODON		
Name	SAT	UNSAT	$\bar{M}_{\text{sat}}$	SAT	UNSAT	$\bar{M}_{\text{sat}}$	SAT	UNSAT	$\bar{M}_{\text{sat}}$
74182	19	45	5.67	1240	0	28	0	0	0
74L85	1	27	1	178	0	20	13	7	13
74283	34	57	5.32	561	0	20	15	5	15
74181	12	43	2.4	691	0	20	4	16	4
c432	10	29	4.7	1 109	0	20	5	15	5
c499	2	118	1	707	0	20	2	12	2
c880	27	86	1.74	12 663	0	20	15	0	15
c1355	36	162	4.1	3 246	0	20	8	3	8
c1908	13	35	1	3 593	4	7	0	2	0
c2670	7	30	7	25	0	19	17	2	17
c3540	38	77	1.86	231	10	10	1	17	1
c5315	0	55	0	1 665	0	20	0	13	0
c6288	8	30	0.27	126	0	2	2	2	2
c7552	7	53	0.64	1 493	3	17	1	17	1
Averaged	15.29	60.50	2.62	1 966.29	1.21	17.36	5.93	7.93	5.93

Table 14: Synthetic systems satisfiability results

ferences were evident in the peak memory usage metric when run on Linux versus Windows. The problem was mostly bypassed by running all but one Java DA on Linux.

6 CONCLUSION

We presented a framework for evaluating and comparing DAs under identical conditions. The framework is general enough to be applied to any system and any kind of DA. The run-time architecture was designed to be as platform independent as possible. We defined a set of metrics that might be of interest when designing a diagnostic algorithm and the framework includes tools to compute the metrics by comparing actual scenarios and diagnostic results.

Using the framework, we have experimented with 13 diagnostic algorithms on 16 systems of various size and synthetic/real-world origin. We have, both manually and programatically, created 1 651 observation scenarios of various complexity. We have designed 10 metrics for measuring diagnostic performance. This has resulted in the execution of 6 484 scenarios with a total duration of more than 169.7 hours and the computation of 84 292 metrics.

We presented the results from our effort to evaluate the performance of a set of diagnostic algorithms on the ADAPT electrical power system testbed, and a set of synthetic circuits. We learned valuable lessons in trying to complete this effort. One major take-away is that there is still a lot of work and discussion needed to determine a common comparison and evaluation framework for the diagnosis community. The other key observation is that no DA was able to be best in a majority of the metrics. This clearly indicates that the selection of DAs would necessarily involve a trade-off analysis between various performance metrics.

The framework presented is by no means a finished product and we expect it to evolve over the years. In the paper, we have identified some of the limitations and expected scope for future expansion. Our sincere hope is that the framework is adopted by growing number of people and applied to a wide variety of physical systems including diagnosis algorithms from several different research communities. The long-term goal is to create a database of performance evaluation results which will allow system designers to choose the appropriate DA for their system given the constraints and metrics in their application.

ACKNOWLEDGEMENTS

We extend our gratitude to Gautam Biswas (Vanderbilt University), Kai Goebel (University Space Research Association), Ole Mengshoel (Carnegie Mellon University), Gregory Provan (University College Cork), Peter Struss (Technical University Munich), Serdar Uckun (PARC), and many others for valuable discussions in establishing the evaluation framework. In addition, we extend our gratitude to David Hall (Stinger Ghaffarian Technologies), David Jensen (Oregon State University), David Nishikawa (NASA), Brian Ricks (University of Texas at Dallas), Adam Sweet (NASA), Michel Wilson (Delft University of Technology), Stephanie Wright (Vanderbilt University), and many others for supporting the work reported here.

This research was supported in part by the National Aeronautics and Space Administration (NASA) Aeronautics Research Mission Directorate (ARMD) Aviation Safety Program (AvSP) Integrated Vehicle Management (IVHM) Project. Additionally, this material is based upon the work supported by NASA under award NNA08CG83C.

This research was supported by PROGRESS, the embedded systems research program of the Dutch organisation for Scientific Research NWO, the Dutch Ministry of Economic Affairs and

the Technology Foundation STW under award DES.07015.

NOMENCLATURE

IN	inputs
OUT	outputs
COMPS	components
V	variables
C	clauses
t_d	first detection
t_i	last isolation
C_s	startup CPU cycles
C	CPU cycles per step
M	memory in use
$\omega^\star$	injected fault
$t_i^\star$	injection of fault i
Ω	candidate diagnoses
$\Omega^\top$	satisfiable candidate diagnoses
W	candidate weights
f	number of all components
n	number of false negatives
N	number of healthy components
$\bar{n}$	number of false positives
$\bar{N}$	number of faulty components
m_{ia}	candidate isolation accuracy
m_{sru}	candidate system repair utility
m_{dru}	candidate diagnosis repair utility
m_{utl}	candidate utility
M_{fd}	scenario fault detection time
M_{fn}	scenario false negative
M_{fp}	scenario false positive
M_{da}	scenario detection accuracy
M_{fi}	scenario fault isolation time
M_{ia}	scenario isolation accuracy
M_{err}	scenario classification errors
M_{utl}	scenario utility
M_{sru}	scenario system repair utility
M_{dru}	scenario diagnosis repair utility
M_{sat}	scenario consistency
M_{cpu}	scenario CPU load
M_{mem}	scenario memory load
$\bar{M}_{\text{fd}}$	system fault detection time
$\bar{M}_{\text{fn}}$	system false negative
$\bar{M}_{\text{fp}}$	system false positive
$\bar{M}_{\text{da}}$	system detection accuracy
$\bar{M}_{\text{fi}}$	system fault isolation time
$\bar{M}_{\text{err}}$	system classification errors
$\bar{M}_{\text{utl}}$	system utility
$\bar{M}_{\text{sru}}$	system system repair utility
$\bar{M}_{\text{dru}}$	system diagnosis repair utility
$\bar{M}_{\text{sat}}$	system consistency
$\bar{M}_{\text{cpu}}$	system CPU load
$\bar{M}_{\text{mem}}$	system memory load

REFERENCES

Bartyś, M., Patton, R., Syfert, M., de las Heras, S., & Quevedo, J. (2006). Introduction to the DAMADICS actuator FDI benchmark study. *Control Engineering Practice*, *14*, 577–596.

Basseville, M., & Nikiforov, I. V. (1993). *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall.

Brglez, F., & Fujiwara, H. (1985). A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In *Proc. ISCAS'85*, pp. 695–698.

Committee E-32, S. A. P. S. H. M. (2008). Health and usage monitoring metrics, monitoring the monitor. Tech. rep. ARP5783.

Davis, M., Logemann, G., & Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, *5*(7), 394–397.

de Freitas, N. (2002). Rao-blackwellised particle filtering for fault diagnosis. In *Proc. AEROCONF'02*, Vol. 4, pp. 1767–1772.

de Kleer, J. (2009). Minimum cardinality candidate generation. In *Proc. DX'09*, pp. 397–402.

de Kleer, J., Mackworth, A., & Reiter, R. (1992). Characterizing diagnoses and systems. *Artificial Intelligence*, *56*(2-3), 197–222.

de Kleer, J., & Williams, B. (1987). Diagnosing multiple faults. *Artificial Intelligence*, *32*(1), 97–130.

DePold, H. R., Rajamani, R., Morrison, W. H., & Pattipati, K. R. (2006). A unified metric for fault detection and isolation in engines. In *Proc. TURBO'06*.

DePold, H. R., Siegel, J., & Hull, J. (2004). Metrics for evaluating the accuracy of diagnostic fault detection systems. In *Proc. TURBO'04*.

Feldman, A., Provan, G., & van Gemund, A. (2007). Interchange formats and automated benchmark model generators for model-based diagnostic inference. In *Proc. DX'07*, pp. 91–98.

Feldman, A., Provan, G., & van Gemund, A. (2008a). Computing minimal diagnoses by greedy stochastic search. In *Proc. AAAI'08*, pp. 911–918.

Feldman, A., Provan, G., & van Gemund, A. (2008b). Computing observation vectors for max-fault min-cardinality diagnoses. In *Proc. AAAI'08*, pp. 911–918.

Feldman, A., Provan, G., & van Gemund, A. (2009). The Lydia approach to combinational model-based diagnosis. In *Proc. DX'09*, pp. 403–408.

Forbus, K., & de Kleer, J. (1993). *Building Problem Solvers*. MIT Press.

Gertler, J. J. (1998). *Fault Detection and Diagnosis in Engineering Systems*. Marcel Dekker.

Grastien, A., & Kan-John, P. (2009). Wizards of Oz – description of the 2009 DXC entry. In *Proc. DX'09*, pp. 409–413.

Heller, U., & Struss, P. (2001). G$^+$DE - the generalized diagnosis engine. In *Proc. DX'01*, pp. 79–86.

Hoyle, C., Mehr, A. F., Tumer, I. Y., & Chen, W. (2007). Cost-benefit quantification of ISHM in aerospace systems. In *Proc. IDETC/CIE'07*.

Iverson, D. L. (2004). Inductive system health monitoring. In *Proc. ICAI'04*.

Izadi-Zamanabadi, R., & Blanke, M. (1999). A ship propulsion system as a benchmark for fault-tolerant control. *Control Engineering Practice*, *7*(2), 227–240.

Karin, L., Lunde, R., & Münker, B. (2006). Model-based failure analysis with RODON. In *Proc. ECAI'06*.

Kavčič, M., & Juričić, Đ. (1997). A prototyping tool for fault tree based process diagnosis. In *Proc. DX'97*, pp. 129–133.

Kurien, J., & Moreno, M. D. R. (2008). Costs and benefits of model-based diagnosis. In *Proc. AEROCONF'08*.

Kurtoglu, T., Mengshoel, O., & Poll, S. (2008). A framework for systematic benchmarking of monitoring and diagnostic systems. In *Proc. PHM'08*.

Kurtoglu, T., Narasimhan, S., Poll, S., Garcia, D., Kuhn, L., de Kleer, J., van Gemund, A., & Feldman, A. (2009). First international diagnosis competition - DXC'09. In *Proc. DX'09*, pp. 383–396.

Larsson, J. E. (1996). Diagnosis based on explicit means-end models. *Artificial Intelligence*, *80*(1).

Lerner, U., Parr, R., Koleer, D., & Biswas, G. (2000). Bayesian fault detection and diagnosis in dynamic systems. In *Proc. AAAI'00*, pp. 531–537.

Mengshoel, O. (2007). Designing resource-bounded reasoners using Bayesian networks: System health monitoring and diagnosis. In *Proc. DX'07*, pp. 330–337.

Metz, C. E. (1978). Basic principles of ROC analysis. *Nuclear Medicine*, *8*(4), 283–298.

Narasimhan, S., & Brownston, L. (2007). HyDE - a general framework for stochastic and hybrid modelbased diagnosis. In *Proc. DX'07*, pp. 162–169.

Orsagh, R. F., Roemer, M. J., Savage, C. J., & Lebold, M. (2002). Development of performance and effectiveness metrics for gas turbine diagnostic technologies. In *Proc. AEROCONF'02*, Vol. 6, pp. 2825–2834.

Poll, S., Patterson-Hine, A., Camisa, J., Garcia, D., Hall, D., Lee, C., Mengshoel, O., Neukom, C., Nishikawa, D., Ossenfort, J., Sweet, A., Yentus, S., Roychoudhury, I., Daigle, M., Biswas, G., & Koutsoukos, X. (2007). Advanced diagnostics and prognostics testbed. In *Proc. DX'07*, pp. 178–185.

Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, *32*(1), 57–95.

Roemer, M., Dzakowic, J., Orsagh, R. F., Byington, C. S., & Vachtsevanos, G. (2005). Validation and verification of prognostic health management technologies. In *Proc. AEROCONF'05*, pp. 3941–3947.

Roychoudhury, I., Biswas, G., & Koutsoukos, X. (2009). Designing distributed diagnosers for complex continuous systems. *IEEE Trans. on Automation Science and Engineering*, *6*(2), 277–290.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education.

Schuster, E. F., & Sype, W. R. (1987). On the negative hypergeometric distribution. *International Journal of Mathematical Education in Science and Technology*, *18*(3), 453–459.

Siddiqi, S., & Huang, J. (2007). Hierarchical diagnosis of multiple faults. In *Proc. IJCAI'07*, pp. 581–586.

Simon, L., Bird, J., Davison, C., Volponi, A., & Iverson, R. E. (2008). Benchmarking gas path diagnostic methods: A public approach. In *Proc. ASME Turbo Expo 2008*.

Sorsa, T., & Koivo, H. (1998). Application of artificial neural networks in process fault diagnosis. *Automatica*, *29*(4), 843–849.

Williams, Z. (2006). Benefits of IVHM: An analytical approach. In *Proc. AEROCONF'06*, pp. 1–9.

Zymnis, A., Boyd, S., & Gorinevsky, D. (2009). Relaxed maximum a posteriori fault identification. *Signal Processing*, *89*(6), 989–999.

A DERIVATIONS OF METRICS

This appendix provides detailed derivation of the formulae for the technical accuracy metrics. In this appendix we use notation of Sec. 3.3 (in particular, recall Fig. 4 and Table 3).

A.1 Classification Errors and Isolation Accuracy

Recall the definition of M_{err} and M_{ia}:

$$M_{\text{err}} = \sum_{\omega \in \Omega} W(\omega)(|\omega \ominus \omega^\star|) \tag{20}$$

$$M_{\text{ia}} = \sum_{\omega \in \Omega} W(\omega)(f - |\omega \ominus \omega^\star|) \tag{21}$$

One can see that M_{ia} and M_{err} are duals, i.e.:

$$\frac{M_{\text{ia}}}{f} + \frac{M_{\text{err}}}{f} = 1 \tag{22}$$

Consider the isolation accuracy (m_{ia}) of a single diagnostic candidate $\omega \in \Omega$:

$$m_{\text{ia}} = f - |\omega \ominus \omega^\star| \tag{23}$$

Eq. 23 defines a plane in the $(n, \bar{n}, m_{\text{ia}})$-space (see Fig 13).

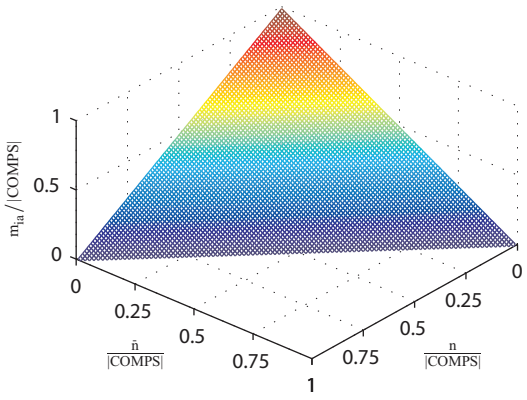

Figure 13: m_{ia} as a function of n and $\bar{n}$

m_{ia} "penalizes" a DA for each misclassified component. As is visible from Fig. 13, the penalty is applied linearly.

The isolation accuracy metric M_{ia} originates in the automotive industry (Committee E-32, 2008). The Aerospace Recommended Practice (ARP) computes the closely related probability of correct classification in the following way. For each component we compute the square confusion matrix. The probability of correct classification is the sum of the main diagonal divided by the total number of classifications (see the referenced ARP (Committee E-32, 2008) for details and examples).

It can be shown that the probability of correct classification, as defined in the above ARP, is equivalent to M_{ia}, if both fault and nominal component modes are used for the computation of the confusion matrices. The probability of correct classification is conditioned on the fault probability while the probability measured by M_{ia} is not. The latter is purely a metric design consideration. The fact that we use nominal modes for computing M_{ia} leads to higher correlation of M_{ia} with the detection accuracy metrics defined later in this section.

If more than one predicted mode vector is reported by a DA, (meaning that the diagnostic output consists of a set of candidate diagnoses), then the isolation accuracy and the classification errors are calculated for each predicted component mode vector and weighted by the candidate probabilities reported by the DA as it is seen in Eq. (20) and Eq. (14). M_{ia} and M_{err} are very useful for single diagnoses but with multiple candidates they are less intuitive. The metric that follows is loosely based on the concept of "repair effort" and partly remedies this problem.

A.2 Utilities

In what follows we show the derivations of the three utility metrics (system repair utility M_{sru}, diagnosis repair utility M_{dru}, and utility M_{utl}).

A.2.1 System Repair Utility

Consider an injected fault $\omega^\star$ ($\omega^\star$ is a set of faulty components) and a diagnostic candidate ω (the set of components the DA considers faulty). The number of truly faulty components that are improperly diagnosed by the diagnostic algorithm as healthy (false negatives) is $n = |\omega^\star \backslash \omega|$ (see Fig. 4). In general a diagnostician has to perform extra work to verify a diagnostic candidate ω, which must be reflected in the system repair utility. We assume that he or she has access to a test oracle that reports if a component c is healthy or faulty.

We first determine what the expected number of tests a diagnostician has to perform to test all components in $\omega^\star \setminus \omega$ (the false negatives) if the diagnostician chooses untested components at random with uniform probability. In the worst case, the diagnostician has to test all the remaining $\text{COMPS} \setminus \omega$ components (the diagnostic algorithm has already determined the state of all components in ω). Consider the average situation. We denote $N = |\text{COMPS} \setminus \omega|$. N is the size of the "population" of components to be tested.

The probability of observing $s-1$ successes (faulty components) in $k+s-1$ trials (i.e., k oracle tests) is given by the direct application of the hypergeometric distribution:

$$p(k, s-1) = \frac{\binom{n}{s-1}\binom{N-n}{k}}{\binom{N}{k+s-1}} \tag{24}$$

The probability $p(k, s)$ of then observing a faulty component in the next oracle test is simply the number of remaining false negatives $n-(s-1)$ divided by the size of the remaining population $(N-(s+k-1))$:

$$p(k, s) = \frac{n-s+1}{N-k-s+1} \tag{25}$$

and the probability of having exactly k oracle faults up to the $s^{\text{-th}}$ test, is then the product of

these two probabilities:

$$p'(k,s,n,N) = \frac{\binom{n}{s-1}\binom{N-n}{k}(n-s+1)}{\binom{N}{k+s-1}(N-k-s+1)} \quad (26)$$

The formula above is the probability mass of the inverse hypergeometric distribution that, in our case, yields the probabilities for testing k healthy components before we find s faulty components out of the population (no repetitions). The expected value $\mathrm{E}'[k]$ of $p'(k,s,n,N)$ (from the definition of a first central moment of a random variable) is:

$$\mathrm{E}'[k] = \sum_{x=0}^{n} x p'(x,s,n,N) \quad (27)$$

Replacing $p'(k,s,n,N)$ in (27) and simplifying gives us the mean of the inverse hypergeometric distribution[2]:

$$\mathrm{E}'[k] = \frac{s(N-n)}{n+1} \quad (28)$$

As we are interested in finding $s = n$ faulty components, the expected value $E'(n,N)$ becomes:

$$\mathrm{E}'[k] = \frac{n(N-n)}{n+1} \quad (29)$$

The expected number of tests $\mathrm{E}[t]$ (as opposed to the expected number of faulty components $\mathrm{E}'[k]$) then becomes:

$$\mathrm{E}[t] = \frac{n(N-n)}{n+1} + n = \frac{n(N+1)}{n+1} \quad (30)$$

The expected number of tests $\mathrm{E}[t]$ is then normalized by the number of components f and flipped alongside the y axis to give the system repair utility:

$$m_{\mathrm{sru}} = 1 - \frac{n(N+1)}{f(n+1)} \quad (31)$$

Plotting the system repair utility m_{sru} against a variable number of false negatives is shown in Fig. 14. One can see that unlike m_{err} which changes linearly, m_{sru} "penalizes" improperly diagnosed components exponentially.

The system repair utility for a set of diagnoses is defined as:

$$M_{\mathrm{sru}} = \sum_{\omega\in\Omega} W(\omega) m_{\mathrm{sru}}(\omega^\star, \omega) \quad (32)$$

where $W(\omega)$ is the weight of a diagnosis ω such that:

$$\sum_{\omega\in\Omega} W(\omega) = 1 \quad (33)$$

All weights $W(\omega)$, $\omega \in \Omega$, are computed by the diagnostic algorithm.

[2]For a detailed derivation of the negative hypergeometric mean, see (Schuster & Sype, 1987).

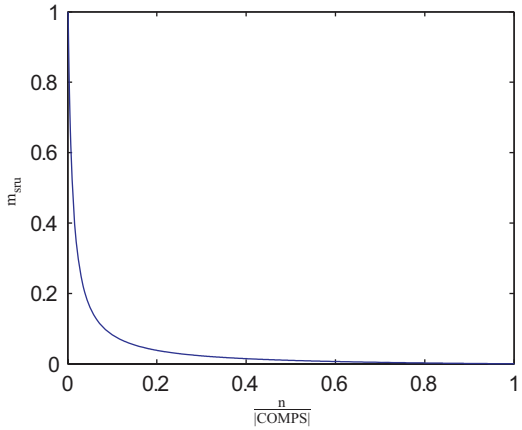

Figure 14: m_{sru} as a function of n

A.3 Diagnosis Repair Utility

Using $\mathrm{E}[t]$ in a metric is not enough as it only captures the effort to "eliminate" (test) all false negatives. The size of the set of false positives is $\bar{n} = |\omega \setminus \omega^\star|$ (see Fig. 4). To find all false positives, the diagnostician has to test in the worst case all components in ω. Hence, the general population is $\bar{N} = |\omega|$. Repeating the argument for $\mathrm{E}[t]$ we determine the expected number of tests for testing all false positives $\mathrm{E}[\bar{t}]$:

$$\mathrm{E}[\bar{t}] = \frac{\bar{n}(\bar{N}+1)}{\bar{n}+1} \quad (34)$$

Similarly, the diagnostic repair utility m_{dru} is the normalized $\mathrm{E}[\bar{t}]$:

$$m_{\mathrm{dru}} = 1 - \frac{\bar{n}(\bar{N}+1)}{f(\bar{n}+1)} \quad (35)$$

The diagnosis repair utility for a set of diagnoses is defined as:

$$M_{\mathrm{dru}} = \sum_{\omega\in\Omega} W(\omega) m_{\mathrm{dru}}(\omega^\star, \omega) \quad (36)$$

A.4 Utility

The utility metric (per candidate) is a combination of m_{sru} and m_{dru}:

$$m_{\mathrm{utl}} = 1 - \frac{\mathrm{E}[t] + \mathrm{E}[\bar{t}]}{f} = \quad (37)$$

$$= 1 - \frac{n(N+1)}{f(n+1)} - \frac{\bar{n}(\bar{N}+1)}{f(\bar{n}+1)} \quad (38)$$

The utility metric (per scenario) is

$$M_{\mathrm{utl}} = \sum_{\omega\in\Omega} W(\omega) m_{\mathrm{utl}}(\omega^\star, \omega) \quad (39)$$

Figure 15 plots m_{utl} for varying numbers of false negatives and false positives in a (symmetric) case where the cardinality of the injected fault is half the number of components. Normally, the number of injected faulty components $|\omega^\star|$ is small

compared to the total number of components f, which leads to an asymmetric m_{utl} plot. In such cases, $\bar{N} \ll N$, hence the role of the false positives is small. In Fig. 15, there is a global optimum $m_{\text{utl}} = 1$ for $n = 0$ and $\bar{n} = 0$, i.e., all components in ω are classified correctly.

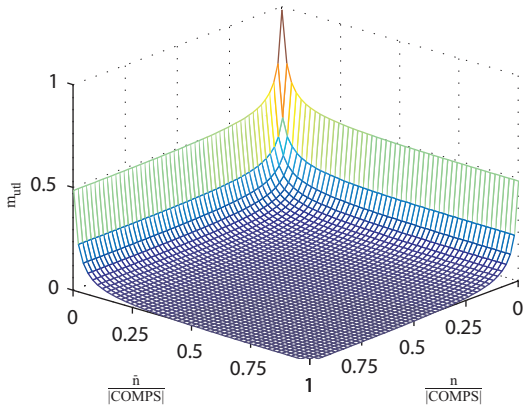

Figure 15: m_{utl} as a function of n and $\bar{n}$

B SYSTEM DESCRIPTION FORMAT

Consider c17, the smallest ISCAS85 circuit (too simple to include in our benchmark). An example system description starts by defining a number of components in the following manner (we have truncated the XML code):

```
<?xml version="1.0" encoding="UTF-8"?>

<systemCatalog ...>
  <systemInstances>
    <systemInstance id="c17" system="c17" />
  </systemInstances>
  <systems>
    <system>
      <systemName>c17</systemName>
      <description>
        The c17 ISCAS85 combina-
tional circuit.
      </description>
      <components>
        <component>
          <name>i1</name>
          <componentType>port</componentType>
        </component>
        <component>
          <name>gate11</name>
          <componentType>nand2</componentType>
        </component>
        <component>
          <name>gate11.o</name>
          <componentType>wire</componentType>
        </component>
              ⋮
```

Part of the topology of c17 is described in the XML excerpt below:

```
<connections>
  <connection>
    <c1>gate10.o</c1><c2>z1</c2>
  </connection>
  <connection>
    <c1>gate10.i1</c1><c2>i1</c2>
  </connection>
  <connection>
    <c1>gate10.i2</c1><c2>i3</c2>
  </connection>
        ⋮
<connections>
```

The component type specifying a circuit breaker and shown next is part of ADAPT-Lite and ADAPT (this component type is referenced, for example, by a component with unique identifier CB180):

```
<componentType xsi:type="circuitBreaker">
  <name>CircuitBreaker4Amp</name>
  <description>
    4 Amp CircuitBreaker
  </description>
  <modesRef>CircuitBreaker</modesRef>
  <rating>4</rating>
</componentType>
```

Another example of a component type is the AC voltage sensor shown below.

```
<componentType xsi:type="sensor">
  <name>ACVoltageSensor</name>
  <description>
    AC voltage sensor.
  </description>
  <modesRef>ScalarSensor</modesRef>
  <sensorValue xsi:type="numberValue">
    <dataType>double</dataType>
    <rangeMin>0</rangeMin>
    <rangeMax>150</rangeMax>
  </sensorValue>
  <engUnits>VAC</engUnits>
</componentType>
```

Below is shown a nand-gate, part of a digital circuit.

```
<componentType>
  <name>nand2</name>
  <description>
    A 2-input logic NAND gate.
  </description>
  <modesRef>gate</modesRef>
</componentType>
```

Finally, we have the modes of a circuit-breaker.

```
<modeGroup>
  <name>CircuitBreaker</name>
  <mode xsi:type="mode">
    <name>Nominal</name>
    <description>
      Transmits current and voltage ...
    </description>
  </mode>
  <mode xsi:type="mode">
    <name>Tripped</name>
    <description>
```

```
      Breaks the circuit and must be ...
    </description>
  </mode>
  <mode xsi:type="faultMode">
    <name>FailedOpen</name>
    <description>
      Trips even though current is ...
    </description>
    <faultSource>Hardware</faultSource>
    <parameters/>
  </mode>
</modeGroup>
```

C MESSAGE FORMATS

Though there are additional message types, the most important messages for the purpose of benchmarking are the sensor data message, command message, and diagnosis message, described below.

C.1 Sensor/Command Data

Sensor data are defined broadly as a map of sensor IDs to sensor values (observations). Sensor values can be of any type; currently the framework allows for integer, real, boolean, and string values. The type of each observation is indicated by the system's XML catalog.

SensorMessage
+timestamp
+sensorValues: Map<sensorIds→sensorValues>

CommandMessage
+timestamp
+commandID: string
+command: commandValue

Table 15: Sensor and command message format

Commandable components contain an additional entry in the system catalog specifying a command ID and command value type (analogous to sensor value type). The command message represents the issuance of a command to the system. In the ADAPT system, for example, the message (EY144_CL, true) signifies that relay EY144 is being commanded to close. "EY144_CL" is the command ID, and "true" is the command value (in this case, a Boolean).

C.2 Diagnosis Result Format

The DA's output (i.e., estimate of the physical status of the system) is standardized to facilitate the generation of common data sets and the calculation of the benchmarking metrics, which are introduced in Sec. 3.3. The resulting diagnosis message is summarized in Table 16 and contains:

timestamp: a value indicating when the diagnosis has been issued by the algorithm.

candidateSet: a candidate fault set is a list of candidates an algorithm reports as a diagnosis. A candidate fault set may include a single candidate with a single or multiple faults; or multiple candidates each with a single or multiple faults. It is assumed that only one candidate in a candidate fault set can represent the system at any given time.

detectionSignal: a Boolean value as to whether the diagnosis system has detected a fault.

isolationSignal: a Boolean value as to whether the diagnosis system has isolated a candidate or a set of candidates.

DiagnosisMessage
+timestamp
+candidateSet: Set <Candidate>
+detectionSignal: Boolean
+isolationSignal: Boolean
+notes: string

Candidate
+faults: Map<componentIds→componentState>
+weight: double

Table 16: Diagnosis message format

In addition, each candidate in the candidate set has an associated weight. Candidate weights are normalized by the framework such that their sum for any given diagnosis is 1.