Second International Diagnostic Competition (DXC'10)

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

[1]Mission Critical Technologies @ NASA Ames Research Center
[2]University of California, Santa Cruz @ NASA Ames Research Center
[3]NASA Ames Research Center
[4]Stinger Ghaffarian Technologies @ NASA Ames Research Center
[5]Palo Alto Research Center
[6]Delft University of Technology

May 7, 2010

Contents

1 Introduction

The Diagnostic Competition (DXC'10) will be the second of a series of international competitions to be hosted yearly at the International Workshop on Principles of Diagnosis (DX). The objectives of the competition are to accelerate research in theories, principles, and computational techniques for monitoring and diagnosis of complex systems, to encourage the development of software platforms that promise more rapid, accessible, and effective maturation of diagnostic technologies, and to provide a forum that can be utilized by algorithm developers to test and validate their technologies on real-world physical systems. The overall goal of this competition is to systematically evaluate diagnostic technologies and to produce comparable performance assessments for different diagnostic methods. (This document extends and adopts the notions of Kurtoglu et al. [2009].)

1.1 Competition Format

Participating diagnostic technologies will be run under same conditions using the same input. The competition consists of multiple tracks (with independent winners). For the first time this year, each track will define a specific use case under which the participating diagnostic algorithms will be evaluated. For each track we will provide a number of standardized specifications of systems used, a library of modular and standardized test scenarios, a test protocol and associated software architecture, and evaluation criteria (i.e. metrics). These metrics will then be computed based on the diagnostic output to determine the final winner(s) in each track.

1.2 Participation Procedures

Those who wish to compete must register at the competition website and must make their tool available to competition organizers in an approved format. Note that participants will not be required to make the source code of their algorithms publicly available. However, competitors should be able to describe how their algorithms work for the benefit of the diagnosis community. Only a Windows or Linux executable file will be required from the participants. Both prerequisites must be met by the deadlines specified in Table 1. Registration and competition information will be posted on http://www.phmsociety.org/events/workshop/dx/10.

Upon request, participants may be given access to the evaluation code that will be used to calculate the performance of their Diagnostic Algorithm. If the DA is accepted for the competition, the participants will be notified accordingly. Winners of each track are expected to come to the conference and discuss their system in person. This will entail giving a short talk during the competition workshop and being available to demonstrate the system and answer questions during the demonstration session. Additionally, all participants will be given an opportunity to submit an optional paper describing their DA and its use. These papers will be subject to review by the DXC'10 organizers and will be published as part of the conference proceedings in a special DXC'10 session. The organizing committee reserves the right to modify the rules of participation and disqualify any participants at its discretion.

1.3 Important Dates & Tentative Schedule

The tentative schedule of the competition is shown in Table 1.

Date	Description
March 8, 2010	Formal announcement of DXC'10
April 7, 2010	Full release of competition information
April 19, 2010	Release of competition data files
May 17, 2010	Release of competition software architecture and API
May 24, 2010	Registration deadline for participation
June 14, 2010	Deadline for submission of 'place holder' version of diagnostic algorithms
August 16, 2010	Deadline for submission of final version of diagnostic algorithms
September 15, 2010	Evaluation of diagnostic algorithms @ NASA Ames
October 13-16, 2010	Present results and winners of challenge @ DX'10

Table 1: Important dates

1.4 Chairs and Organizing Committee

Sriram Narasimhan (UC Santa Cruz at NASA Ames) and Johan de Kleer (PARC) will be co-chairs of DXC'10. Tolga Kurtoglu (MCT at NASA Ames) will be the organizing committee chair. Scott Poll (NASA Ames) and David Garcia (SGT at NASA Ames), Alexander Feldman (Delft University of Technology), and Lukas Kuhn (PARC) are the other members of the organizing committee.

1.5 Acknowledgments

We extend our gratitude to Arjan van Gemund (Delft University of Technology), Gregory Provan (University College Cork), Peter Struss (Technical University Munich), Gautam Biswas (Vanderbilt University), Serdar Uckun (PARC), the DX'10 organizers and many others for valuable discussions, criticism and help.

2 Diagnostic Problem Descriptions and Use Cases

The second competition will consist of three tracks: two industrial and one synthetic. The two industrial tracks concern one system (ADAPT). Each track will define one or more diagnostic scenarios, a specific diagnostic use case, and an associated operational use. Participating DAs may compete in one or more tracks (cf. Table 2).

Name	System Name	Operational Scenario	Diagnostic Use Case
`industrial 1`	ADAPT-Lite	Single-string UAS Mission	Abort Recommendation
`industrial 2`	ADAPT	Redundant Systems UAS Mission	Fault Recovery Recommendation
`synthetic`	ISCAS-85	Digital System Maintenance	Troubleshooting Support

Table 2: Diagnostic problems in the DXC'10 competition

2.1 Industrial Tracks

For DXC'10 we will use the Electrical Power System (EPS) testbed of the ADAPT lab at NASA Ames Research Center for the industrial track system. 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.

The scope of the EPS testbed used for each industrial track diagnostic problem is shown in Fig. 1 and Fig. 2. We have defined two systems from the same physical testbed, ADAPT-Lite and ADAPT.

ADAPT-Lite ADAPT-Lite includes a single battery and multiple loads as shown in the schematic (Fig. 2). 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. ADAPT-Lite will include examples of incipient and intermittent faults.

ADAPT ADAPT includes all batteries and loads in the EPS. The system will start with all relays open and commanded mode changes will bring the system to one of many predefined configurations in accordance with its use case. 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 will be injected in ADAPT.

For both ADAPT and ADAPT-Lite, scenario data will be presented at three different rates: 1, 2, and 10 Hz respectively. The details of sensor data rates will be provided as part of the system documentation.

2.1.1 Diagnostic Problem I: Single-string UAS Mission

The first diagnostic problem will use the ADAPT-Lite system and mimic its use in a single-string UAS (Unmanned Aircraft System) mission. The primary objective of the diagnostic algorithm in this operational scenario is to provide decision support to mission control by making an "Abort" recommendation. "Abort" is defined as aborting the mission and landing the UAS vehicle.

This problem has a non-redundant power configuration of the EPS that supports mission and vehicle critical loads. There is only one path from one power source to two AC loads and one DC load. There are two possible recommendations for the nominal and single-fault scenarios in this diagnostic problem: 'none', and 'abort'. The correct recommendation depends on the injected failure mode and, for certain failure modes, the fault parameters. To build a successful diagnostic system for this problem, all failure modes of respective components must be modeled. However, only a subset of system failure modes/magnitudes will lead to the abort recommendation. Other failure modes/parameters may produce degraded performance or loss of non-critical functionality. Those failure modes/parameters that require an abort recommendation will be provided to the participants as part of the system documentation.

The performance of a diagnostic algorithm in this operational scenario will be judged based on the correctness of the abort recommendations made by the algorithm. There will be consequential costs associated with incorrect or missed abort recommendations. In case of the former (when the DA issues a abort recommendation when it was not needed), the mission value will be assumed to be lost. In case of the latter (when the DA fails to issue an abort recommendation when it was needed), both the mission value and vehicle value will be assumed to be lost. These cost measures will be provided to participants as part of the system documentation. However, this information will also be incorporated as part of the DXC framework. DAs may query the framework with a diagnosis and receive the correct recommendation for that diagnosis. Alternatively, DAs may query the framework with a diagnosis and a recommendation and receive the corresponding cost. By including this capability in the framework a DA will only be responsible for managing ambiguities in its diagnosis and coming up with the final recommendation.

2.1.2 Diagnostic Problem II: Redundant Systems UAS Mission

The second diagnostic problem will use the ADAPT system and mimic its use in a redundant systems UAS mission. The primary objective of the diagnostic algorithm in this operational scenario is to provide decision

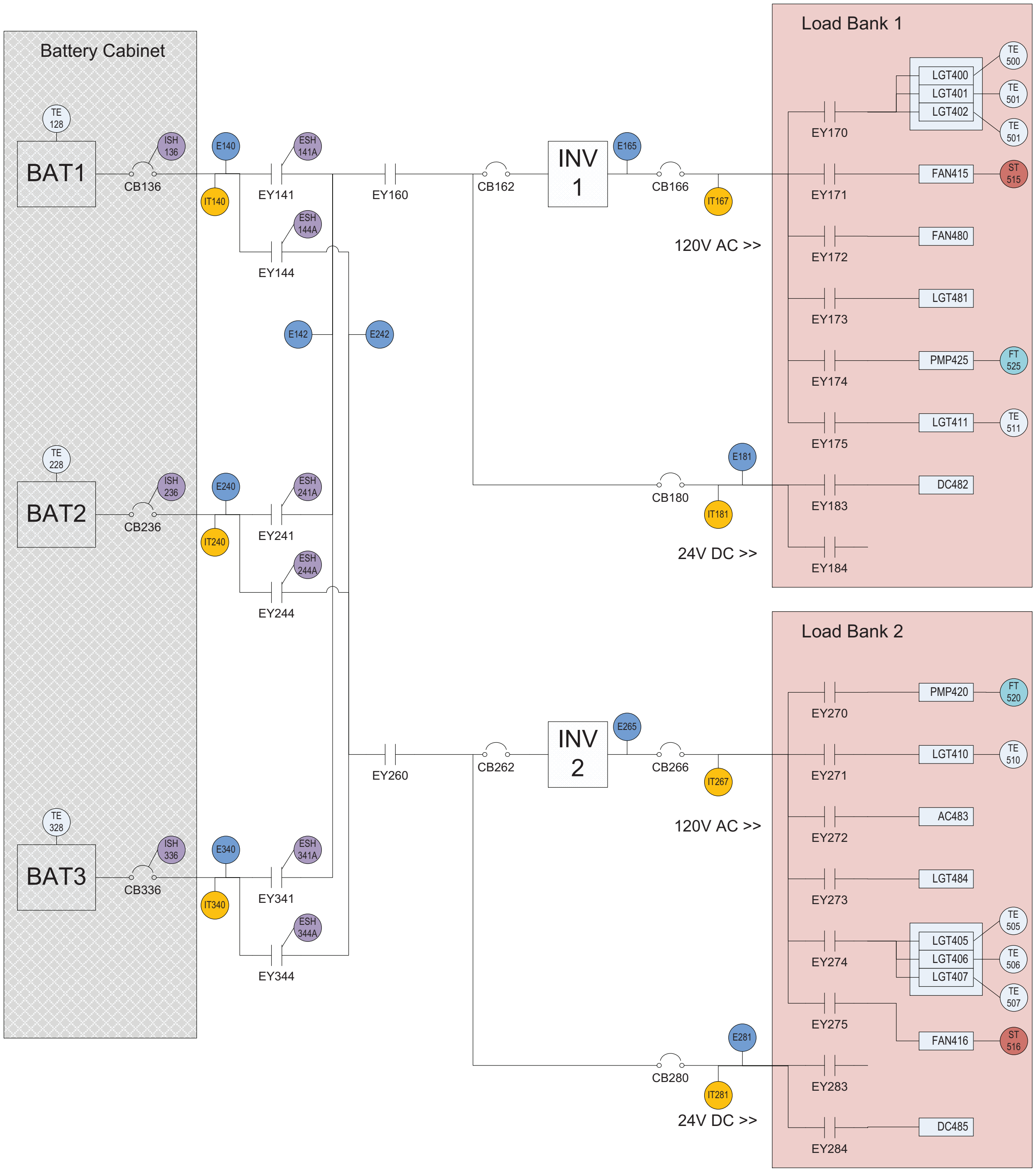

Figure 1: ADAPT EPS

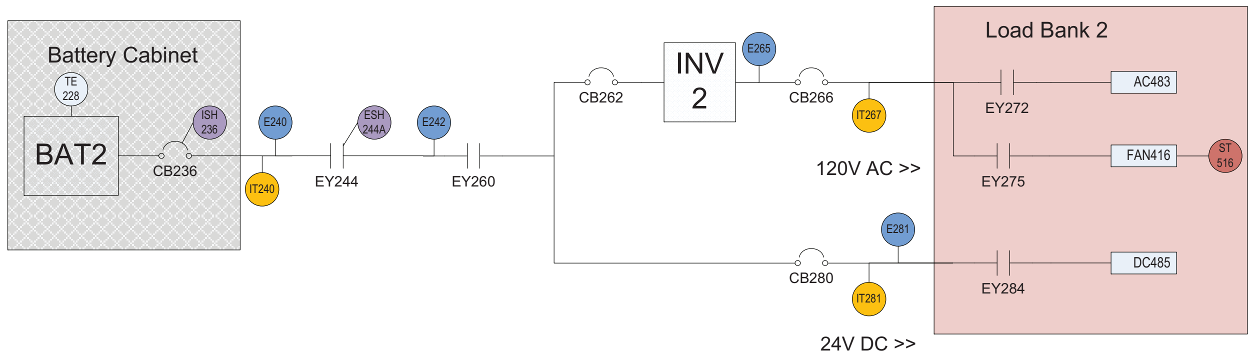

Figure 2: ADAPT-Lite

support to mission control by making recommendations for fault recovery actions.

This problem has redundant power configurations of the EPS system that support mission and vehicle critical loads. There are multiple possible paths from power sources to the loads. Some loads are critical to vehicle operations, some are critical for mission success, and others are considered non-critical. For example, a vehicle critical load may be an avionics computer while a mission critical load may be a sensor payload such as an IR camera. Nominal configurations for this problem provide power to five critical loads (three on one load bank, two on the other) and four non-critical loads (two on each load bank). Furthermore, there are redundant loads located on the opposite load bank for each of the five critical loads; i.e., a critical load on load bank 1 has a redundant load on load bank 2 with the same functionality. Only one of two redundant loads will be on in any given scenario and will remain on for the duration of a nominal scenario while the non-critical loads may be turned on and off. The system documentation will specify the possible configurations. Diagnostic Problem II will include only abrupt, persistent single and multiple faults.

The performance of a diagnostic algorithm in this operational scenario will be judged based on the correctness of the fault recovery recommendations made by the DA. There are several possible recovery actions defined for this problem and more than one action may be required to recover functionality lost by the fault(s) injected into the scenario. There will be consequential costs associated with incorrect or missed recovery action recommendations which lead to the loss of the mission or vehicle. It is also possible that functionality is restored but with unnecessary or less desirable recovery actions, which consequently leads to a higher cost. The cost measures for all possible recovery actions for injected faults will be provided to participants as part of the system documentation. However, this information will also be incorporated as part of the DXC framework. DAs may query the framework with a diagnosis and receive the lowest cost action(s) for that diagnosis. Alternatively, DAs may query the framework with a diagnosis and specific action(s) and receive the corresponding cost. By including this capability in the framework a DA will only be responsible for managing ambiguities in its diagnosis and coming up with the final recovery action(s). Note that this problem does not include active diagnosis and no actions will actually be taken during the scenario.

2.2 Synthetic Track

The synthetic track consists of models of the 74XXX/ISCAS85 combinational circuits. We have chosen models of digital circuits because:

- they are trivial to model, thus facilitating easier comparison of the algorithmic properties of the DAs;
- they are variable in size (74XXX/ISCAS85 circuits have between 19 and 3 512 gates);
- they vary in architecture (ALUs, SEC/DEC, multiplier, adder);
- many electrical, mechanical, hydraulic, etc. systems expose properties similar to 74XXX/ISCAS85.

The synthetic track assumes weak-fault models. A significant challenge for the participating DAs is that some observations lead to minimal-cardinality multiple-fault diagnoses of very high cardinality. Furthermore, in some cases, the number of minimal-cardinality diagnoses in a minimal-cardinality ambiguity group is very large, e.g., tens of thousands of diagnoses. This years DXC'10 addresses the problem of decreasing the diagnostic uncertainty in cases with high diagnostic ambiguity, i.e., the competition includes probing.

For each system the DA will be supplied a list of actions (measurements). Each measurment is a pair $\langle v, c \rangle$ where v is an internal variable and c is the cost of measuring this variable. This year we have the following simplifying assumptions:

- c is always 1;
- all system variables are constant during the diagnosis session;
- measurements are always correct, instantaneous and successful;
- there is always an initial observation that leads to a minimal-cardinality diagnosis of cardinality;

2.2.1 Diagnostic Problem III: System Maintenance

The third diagnostic problem uses 74XXX/ISCAS85 systems and mimic their use for system maintenance. Consider the case in which a technician has to repair a malfunctioning system, for example, a copier. The technician has full access to a model and the artifact itself, observes a subset of all inputs and outputs "for free" (e.g., the copier produces a blank page instead of a page with text, and there is no error displayed on the front panel) and can perform zero or more measurements after the initial observation. A measurement can be lifting the platen cover and observing a hidden indicator or a measurement can include replacing a component and observing the correct functioning of the device afterwards. Each measurement has a cost (technician time and pay rate, transportation cost, etc.) and false replacements, i.e., replacements of healthy components, incur loss as well.

At the end of the maintenance case, the diagnostician will have repaired the fault, i.e., there will be no faulty components in the system. We can quantify the performance of a DA by summing the weighted cost of (1) computation, (2) measurement and experimentation, and (3) diagnostic precision at the final diagnosis. There are more details on metrics in Sec. 4.

2.3 What is different in this year's competition?

In last year's competition (DXC'09), we demonstrated an end-to-end implementation of DXF and created a foundation for future usage of the framework (`http://www.dx-competition.org/`). There were also some simplifying assumptions made last year, which we are beginning to address starting with this year's competition. These newly added challenges include:

- introduction of use cases and newly defined metrics;
- introduction of reduced sensor sets to monitor the system;
- introduction of additional fault classes including simple incipient and intermittent faults;
- introduction of probing capability [de Kleer and Williams, 1987] that allows the DAs to query select system data;
- introduction of data related challenges including process noise and frame drop-outs;
- introduction of multi rate data acquisition capability.

3 Participant Support: What will we provide?

The DXC'10 competition will use a framework called DXF [Kurtoglu et al., 2009] that allows systematic comparison and evaluation of diagnostic algorithms under identical experimental conditions. The process to set up the framework in order to perform comparison/evaluation of a selected set of diagnostic algorithms on specific physical system will be 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 API (described later in this section) in order to receive sensor data and send the diagnosis results.
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.

3.1 XML System Descriptions

As described above, the participants will be asked to model and diagnose a system. A system consists of interconnected components, and each component is an instance of a given component type (e.g., a relay, a valve, an and-gate, a wire).

Name	Description
System Name	Unique identifier
Artifact Description	Brief text summary of the system and pointers to documentation, forums, mailing lists, and other resources
Component Catalog	List of component identifiers, with reference to component type and commands that affect the component
Interconnection Diagram	Each node of the graph contains a component identifier/instance identifier pair, and there is an edge for any two (physically) connected components

Table 3: System description data

XML system descriptions will be provided to describe the functioning of the system. Major system description data is listed in Table 3. Beyond being formatted in XML, they are not formalized. This is to prevent biasing the modeling choices. We recognize the fact that *any* artifact description (e.g., graphical, programmatic, etc.) contains certain biasing towards a modeling approach, but the above applies not only to the description of the artifact but to the artifact itself.

Almost any diagnostic technology today uses some kind of graph-like structure for describing the system structure, hence we will provide graphs of the physical connectivity of the system. This graph is not annotated: for example there is no directional information, etc. The latter kind of data can be extracted from the repository of documents describing the system, if available.

This is an example XML system description:

```
<system>
  <systemName>polycell</systemName>
  <description>
    A familiar circuit. Contact: dekleer@parc.com. Publications: [dW87]
  </description>

  <components>
     <component>
        <name>A</name>
        <type>PROBE</type>
     </component>
     <component>
        <name>M1</name>
        <type>MUL</type>
     </component>
     <component>
        <name>A1</name>
        <type>ADD</type>
     </component>
  </components>

  <connections>
    <connection>
      <c1>A</c1>
      <c1>M1</c1>
    </connection>
    <connection>
      <c1>M1</c1>
      <c1>A1</c1>
    </connection>
  </connections>
</system>
```

3.1.1 XML Component Type Descriptions

Next we have to provide specifications for all component types mentioned in the system description as shown in Table 4. Consider the “CircuitBreaker” ADAPT component type (referenced, for example, by the component with unique ID `CB180`):

```
<componentType xsi:type="circuitBreaker">
  <name>CircuitBreaker4Amp</name>
  <description>
    4 Amp CircuitBreaker. http://adapt.nasa.gov/ Contact: scott.poll@nasa.gov.
  </description>
  <modesRef>CircuitBreaker</modesRef>
  <rating>4</rating>
</componentType>
```

or an “ACVoltageSensor” ADAPT component type:

```
<componentType xsi:type="sensor">
  <name>ACVoltageSensor</name>
  <description>AC voltage sensor.</description>
  <modesRef>ScalarSensor</modesRef>
  <datatype>double</datatype>
  <engUnits>VAC</engUnits>
  <rangeMin>0</rangeMin>
  <rangeMax>150</rangeMax>
  <sampleRate>2</sampleRate>
</componentType>
```

As a part of a more abstract example we can consider a description of an and-gate, part of a digital circuit:

```
<componentType>
  <name>AND2</name>
  <description>
    AND gate.
  </description>
  <modesRef>AndGate</modesRef>
</componentType>
```

3.1.2 XML Mode Catalog

Component operating modes are organized by Mode Groups. More than one component can refer to the same group. The Mode Group format is described in Table 5.
Consider the allowable modes for the CircuitBreaker component from the preceding section:

Name	Description
Component Type Name	Unique identifier
Component Description	Brief summary of the component type and pointers to documentation, forums, and other resources
Modes	Reference to a mode group (Table 5)
Component-Specific Info	Examples: sensor min/max, load wattage, circuit breaker rating

Table 4: Component description data

Name	Description
Modes Group Name	Unique identifier for each mode class
Mode Names	Names of the possible modes
Mode Descriptions	Text descriptions
Mode Parameters	Parameters of each failure mode (if any)

Table 5: Mode group description

```
<modeClass>
  <name>CircuitBreaker</name>
  <mode>
    <name>Nominal</name>
    <description>
      Transmits current and voltage. Trips when current exceeds threshold.
    </description>
  </mode>
  <mode>
    <name>Tripped</name>
    <description>
      Breaks the circuit and must be manually reset.
    </description>
  </mode>
  <mode xsi:type="faultMode">
    <name>FailedOpen</name>
    <description>
      Trips even though current is below threshold.
    </description>
  </mode>
</modeClass>
```

3.1.3 Communications Regarding System Descriptions

Participants may ask competition organizers questions about the artifacts and their components. The organizers will decide whether or not to answer these questions. The only condition is that *all* the dialog will become available to *all* the participants in the competition via the competition website.

3.2 System Data Files

In addition to system description documents, the participants will be provided sample data for all diagnostic problems. Data will be provided for nominal runs and runs with faults injected in three formats: tab delimited .txt files, tab delimited .scn files, and Matlab .mat files. The .scn files are scenario files in the format read by the API, which will be used by diagnosis algorithms to access commands to and sensor data from the systems.

3.3 Software Architecture

The DXC effort includes the development of a software framework for running and evaluating diagnostic algorithms. The framework has been designed with the following considerations in mind: (1) the overhead of interfacing existing diagnostic algorithms should be reduced by supplying minimalistic APIs, (2) inter-platform portability should be provided by allowing clients to interface C++ and Java API or by implementing a simple ASCII based TCP messaging protocol, and (3) the framework should be realistic by reflecting industrial practices and needs.

To facilitate algorithm development and testing, we will provide binary packages for Windows and Linux. All Java and C++ source code will be made available. Allegro CommonLisp versions of the API are available on request. Our framework will contain a very simple diagnostic algorithm and a few examples.

3.3.1 Software Components

Figure 3 shows an overview of the DXC software components and the primary information flows. As we have already mentioned, all communication is ASCII based and all the modules communicate via TCP ports by using a simple message-based protocol which we describe in more detail below.

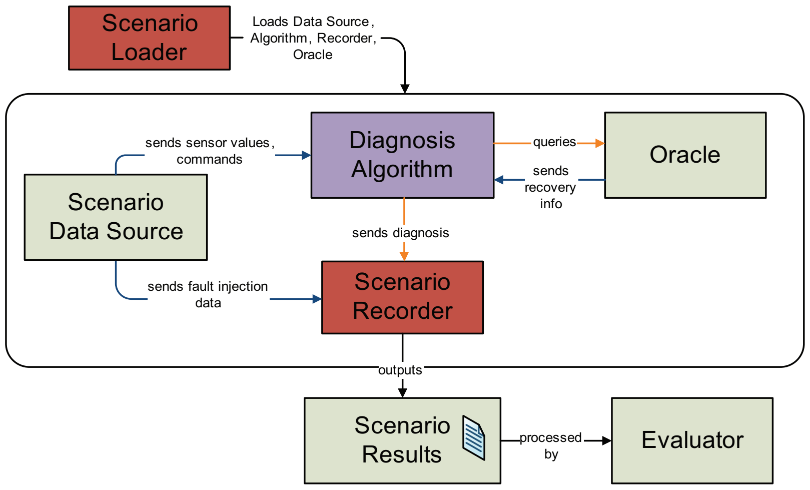

Figure 3: Architecture overview of the DXC software framework

Next, we provide a brief description of each software component.

Scenario Loader: Executes the Scenario Data Source, Recorder, and Diagnostic Algorithm. Ensures system stability and clean-up upon scenario completion. This is the main entry point for performing a diagnostic experiment.

Scenario Data Source: 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[1], and fault injection information (to be sent exclusively to the scenario recorder). The Scenario Data Source will publish data following a wall-clock schedule specified by timestamps in the scenario files.

Scenario Recorder: Receives fault injection data and diagnosis data, and compiles it into a Scenario Results File. The results file contains a number of time-series which will be described below. These time-series are used by the Evaluation module for scoring and can be supplied to the participants for detailed analysis of the algorithmic performance.

The Scenario Recorder is the main timing authority, i.e., it timestamps each message upon arrival before recording it to the Scenario Results File.

Diagnostic Algorithm: A Windows or Linux executable (or Java bytecode in an executable .jar file) contained in a single directory, requiring no compilation, installation, or external libraries except the sockets API and its dependencies. Receives sensor data and command data, performs diagnosis, and sends diagnosis results back.

Diagnostic Oracle: The diagnostic oracle provides a querying capability to the DAs in one of two ways: 1) It takes a diagnostic output produced by a DA and returns the lowest cost action(s) associated with the provided

[1]Note that the majority of classical MBD literature does not discern commands from observations.

diagnosis, or 2) It takes a diagnostic output and specific actions produced by a DA and returns the corresponding cost. Only used in the industrial tracks.

Evaluator: Takes Scenario Results File and applies metrics to evaluate Diagnosis Algorithm performance. The metrics and evaluation procedures are detailed in Sec. 4.

3.3.2 Messaging Datatype Specifications

This section defines the data types for the messages that will be sent to and from a DA. The DXC API and TCP/IP interfaces conform to these data types. The DXF data types are shown in Fig. 4. The two new data types in DXC'10 are `RecoveryData` and `RecoveryAction`.

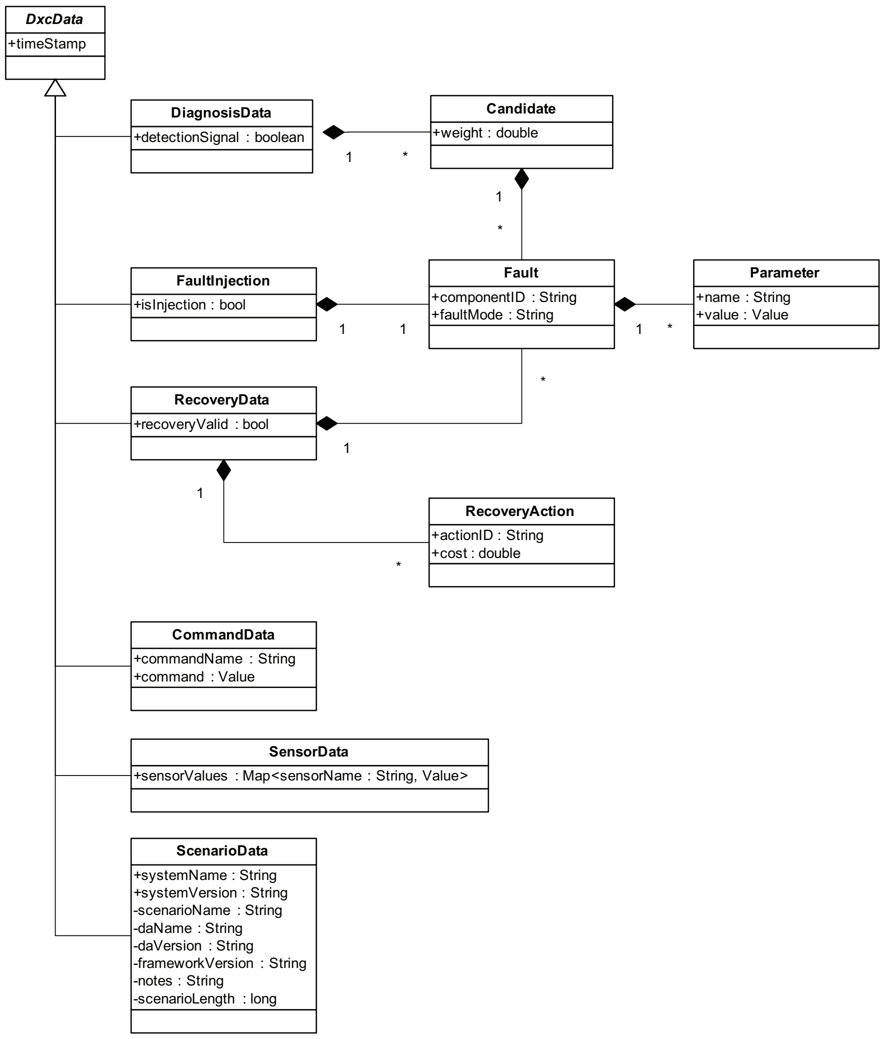

Figure 4: DXF data types

3.3.3 Diagnostic Session Overview

Figure 5 shows a diagnostic session sequence diagram. Note the introduction of the cost oracle and the messages with which a DA requests recovery information/costs and recommends an abort.

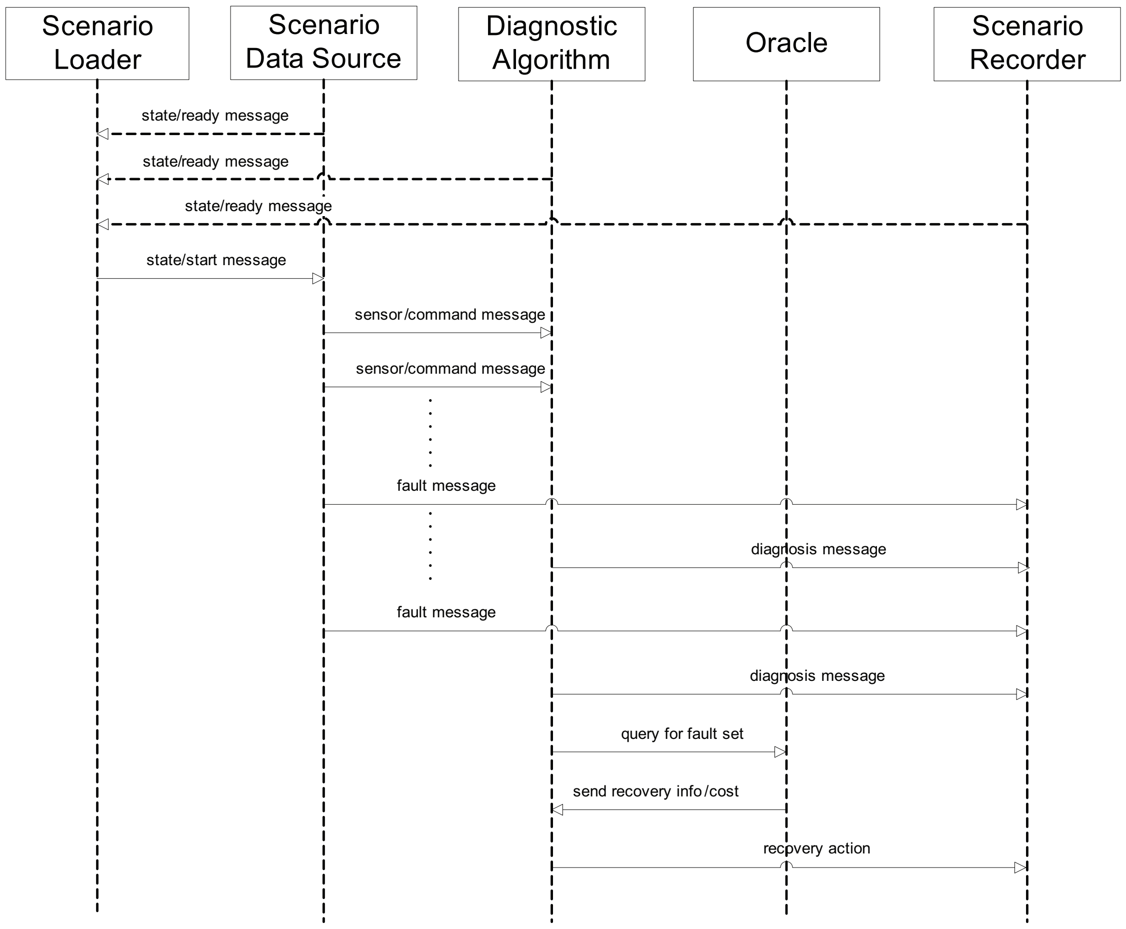

Figure 5: Diagnostic session sequence diagram

3.4 Use Case Specific Operational Information

The participants will be given a detailed description of systems concept of operations for each diagnostic use case and all associated data and information. This information for the first use case (i.e., Single-string UAS Mission) will include the mapping between system failure modes to appropriate abort recommendations. In addition, any critical and noncritical loads for the electrical power system will be described. Finally, cost measures for correct/incorrect abort recommendations will be detailed. Similarly, for the second use case (i.e., Redundant Systems UAS Mission) the mapping between system failure modes and appropriate recovery action recommendations will be provided along with the descriptions of potential system operational modes, critical and noncritical load configurations for the electrical power system. Finally, cost measures for correct/incorrect recovery action recommendations will be detailed.

4 Scoring the Diagnostic Algorithm Performance

We will compute all DXC'09 metrics [Kurtoglu et al., 2009] and, as of this year, decision cost ($M_{\rm dc}$) in the two industrial tracks and active testing utility ($M_{\rm atu}$) in the synthetic track. Note that only $M_{\rm dc}$ and $M_{\rm fi}$ will be used for ranking the DAs in the industrial tracks. In the synthetic tracks, similar to DXC'09, we will use $M_{\rm atu}$, $M_{\rm utl}$, $M_{\rm cpu}$, and $M_{\rm mem}$.

4.1 Performance Metrics

We next show how $M_{\rm dc}$ is computed from the recommendation of a DA and the vehicle, mission, and safing costs.

4.1.1 Metric(s) for Diagnostic Problem I: Single-string UAS Mission

The metrics used to evaluate the DA's performance will be based on the costs incurred if the DA's recommendation is accepted. The two main categories of costs are cost of losing the vehicle and cost of not completing the mission objectives. In this use case the DA is only responsible for deciding if a mission should be aborted or not. Hence there are 4 outcomes (2 answers from the DA versus 2 actual situations). Let the cost of losing the mission be c_{mission}, and the cost of losing the vehicle be c_{vehicle}. The following table computes the costs incurred in each of the 4 possible outcomes.

Actual Situation / DA Recommendation	Abort	Non-Abort
Abort	0	c_{mission}
Non-Abort	$c_{\text{vehicle}} + c_{\text{mission}}$	0

Table 6: Decision costs (M_{dc})

The cost incurred over multiple scenarios is added to arrive at the cumulative cost incurred in following a DA's recommendations. If more than one DA has the least cost incurred then a tie breaker metric will be used based on a single and/or combination of previous metrics defined for DXC'09 DXC'09 metrics [Kurtoglu et al., 2009].

4.1.2 Metric(s) for Diagnostic Problem II: Redundant Systems UAS Mission

For this use case the metrics used are the similar to the costs incurred metric in the Diagnostic Problem I. There are costs associated with losing the mission (c_{mission}) and losing the vehicle (c_{vehicle}). There are also costs associated with each recovery action. For each fault (or faults in multiple fault scenarios) there might be more than one set of recovery actions that mitigate the effects of the fault(s). To give an example we first describe the system configuration in this use case. The EPS provides power to five critical load functions to the UAS, four AC and one DC; the EPS also provides power to four AC loads which are deemed non-critical. The system has passive redundancy for the five critical functions such that there are two identical loads for each critical load function, one attached to load bank 1 and the other attached to load bank 2. Only one or the other load is on at any given time to provide the critical function. For example, suppose that relay EY170 is closed and powers a critical load; there is an identical, unpowered load downstream of open relay EY274. Similarly for loads downstream of relays EY171 (closed) and EY275 (open). Next suppose that battery 1 is connected to load bank 1, battery 2 is connected to load bank 2, and battery 1 subsequently fails. In this scenario, power to the critical loads downstream of EY170 and EY171 is lost. What are the possible recovery actions? One option is to turn on the redundant loads attached to the other load bank by issuing two commands, close relays EY274 and EY275. Another option is to connect battery 3 to load bank 1 by closing relay EY341, which is the preferred option because it has fewer reconfiguration commands and maintains load redundancy. Consequently, this recovery option will have lower cost. If a DA recommends only to close relay EY274, the effects of the fault are not mitigated and there is an additional cost of losing the mission (and vehicle, depending on the lost function). As this example shows, when the DA recommends a set of recovery actions for Diagnostic Problem II it automatically incurs the cost of performing those actions. In addition, it may incur the costs of losing the mission or losing the mission and vehicle if the recovery actions do not mitigate the effects of the faults.

As in Diagnostic Problem I, the tie-breaker will be based on a single and/or combination of previous metrics defined for DXC'09 DXC'09 metrics [Kurtoglu et al., 2009].

4.1.3 Metric(s) for Diagnostic Problem III: Digital System Maintenance

Active Testing Utility (M_{atu}): Let S be the set of all sensors $S = \{s_1, s_2, \ldots, s_n\}$ queried data to a DA and $c(s)$ ($s \in S$) be the cost of obtaining the sensor reading for sensor s. Let the cost of testing whether component d is working correctly be $o(d)$ (for simplicity $o(d) = 1$ this year). The active testing utility metric (per scenario) is then defined as follows:

$$M_{\text{atu}} = \sum_{s \in S} c(s) + o(d)E(t) \tag{1}$$

Where $E(t)$ represents the expected number of component tests required to identify all the faulty components as defined in the formulation of M_{utl}. Thus M_{atu} measures the expected cost of isolating the faulty components. If the

DA isolates the fault correctly given the measurements, then $E(t) = 0$. On the other hand, if a DA renders all possible diagnoses, then $E(t)$ will be large. A DA would clearly try to minimize $M_{\rm atu}$. Our proposal is to use $M_{\rm atu}$, $M_{\rm utl}$, $M_{\rm cpu}$, and $M_{\rm mem}$ as diagnostic metrics in the synthetic track of DXC'10. The probabilities of component failures in the scenario set could have a significant influence on $M_{\rm atu}$. For DXC'10 we made the two simplifying assumptions that all components fault with equal liklihood and that the cardinality of injected fault is equal to the cardinality of the initial ambiguity group (as computed by a perfect algorithm). In this case, the probability of component failure does not influence $M_{\rm atu}$.

4.2 Determination of Winners

The aforementioned cost metrics will be used as the criteria for the ranking of the competing diagnostic algorithms. In cases where two or more diagnostic algorithms score the same in for a given problem, tie breaker metrics will be introduced.

The organizing committee has the right to disqualify any engine which exploits the scoring mechanism to its advantage.

References

Johan de Kleer and Brian Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

Tolga Kurtoglu, Sriram Narasimhan, Scott Poll, David Garcia, Lukas Kuhn, Johan de Kleer, Arjan van Gemund, and Alexander Feldman. First international diagnosis competition - DXC'09. In *Proc. DX'09*, pages 383–396, 2009.