Towards a Framework for Evaluating and Comparing Diagnosis Algorithms

Tolga Kurtoglu*, Sriram Narasimhan, Scott Poll***, David Garcia****, Lukas Kuhn†, Johan de Kleer†, Arjan van Gemund‡, Alexander Feldman†,‡**

** Mission Critical Technologies @ NASA Ames Research Center*
*** University of California, Santa Cruz @ NASA Ames Research Center*
**** NASA Ames Research Center*
***** Stinger Ghaffarian Technologies @ NASA Ames Research Center*
† Palo Alto Research Center
‡ Delft University of Technology

Abstract: Diagnostic inference involves the detection of anomalous system behavior and the identification of its cause, possibly down to a failed unit or to a parameter of a failed unit. Traditional approaches to solving this problem include expert/rule-based, model-based, and data-driven methods. Each approach (and various techniques within each approach) use different representations of the knowledge required to perform the diagnosis. The sensor data is expected to be combined with these internal representations to produce the diagnosis result. In spite of the availability of various diagnosis technologies, there have been only minimal efforts to develop a standardized software framework to run, evaluate, and compare different diagnosis technologies on the same system. This paper presents a framework that defines a standardized representation of the system knowledge, the sensor data, and the form of the diagnosis results – and provides a run-time architecture that can execute diagnosis algorithms, send sensor data to the algorithms at appropriate time steps from a variety of sources (including the actual physical system), and collect resulting diagnoses. We also define a set of metrics that can be used to evaluate and compare the performance of the algorithms, and provide software to calculate the metrics.

1. INTRODUCTION

Fault Diagnosis in physical systems involves the detection of anomalous system behavior and the identification of its cause. Key steps in the 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?). Expert knowledge and prior know-how about the system, models describing the behavior of the system, and sensor data from system during actual operation are used to develop diagnostic inference algorithms. This problem is non-trivial for a variety of reasons including:

- incorrect and/or insufficient knowledge about system behavior
- limited observability
- presence of many different types of faults (system/supervisor/actuator/sensor faults, additive/multiplicative faults, abrupt/incipient faults, persistent/intermittent faults)
- non-local and delayed effect of faults due to dynamic nature of system behavior
- 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 typical 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 (Kostelezky, 1990) and fault trees (Kavcic and Juricic, 1997).
- Model-based Systems – These approaches use an explicit model of the system configuration and behavior to guide the diagnostic inference. Some examples are “FDI” methods (Gertler, 1998), statistical methods (Basseville and Nikorov, 1993), “AI” methods (Hamscher *et al.*, 1992).
- Data-driven Systems – These approaches use only 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 IMS (Iverson, 2004), Neural Networks (Sorsa and Koivo, 1998), etc.
- Stochastic Method – These approaches treat the diagnosis problem as a belief state estimation problem. Some examples are Bayesian Networks (Lerner et al., 2000), Particle Filters (de Freitas, 2001), etc.

Despite the development of such a variety of notations, techniques, and algorithms, efforts to evaluate and compare the different *diagnosis algorithms* (DA) have been minimal (discussed in Section 2). One of the major deterrents is the lack of a common framework for evaluating and comparing diagnosis 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 diagnosis technologies
- Provide a forum that can be utilized by algorithm developers to test and validate their technologies
- Systematically evaluate different diagnosis technologies by producing comparable performance assessments

Such a framework would require the following:

- Define a standard representation format for the system description, sensor data, and diagnosis result
- Develop 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.
- Define 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. Section 2 presents the motivation for this effort and some related work in other problem domains like planning and prognosis. Section 3 presents our framework, and describes each component in detail. Section 4 lists the assumptions that we had to make and the limitations of this framework. Finally, Section 5 presents our conclusions and the future of the proposed framework.

The framework discussed in this paper, examples, XML schemas, and other related materials can be downloaded from the DXC website[1].

2. MOTIVATION & RELATED WORK

The DX community meets every year at the International Workshop on the Principles of Diagnosis[2] to discuss the latest developments in the field of model-based diagnosis. One common point of discussion at every meeting concerns the establishment of a means to evaluate and compare diagnosis algorithms. The ISCAS85 digital circuit benchmarks became an informal test suite for one subset of the MBD community, but here was no agreement on either the form of the benchmark or the metrics. At DX'06 the community decided to establish a common set of diagnostic benchmarks – but that effort did not take root. When NASA Ames Research Center presented an Electrical Power System hardware testbed (ADAPT) at the Workshop in 2007 (Poll *et al.*, 2007), the system and the run-time architecture it used to execute DAs was well received. NASA was encouraged to generalize this framework. In searching for similar frameworks and talking to people from other diagnosis communities, we realized that there was a need for a common framework to execute different DAs and evaluate and compare their performance.

The use of benchmark models and algorithms has played an important role in advancing the state-of-the-art in fields such as SAT, planning, multi-agent systems, and recently prognostics.

Beginning in 1998, the international planning community has held a biennial event to support the direct comparison of planning systems on a changing collection of benchmark planning problems. At the time the organizers had several objectives: to allow meaningful comparison of programs, to provide an indication of the overall progress in the field, to present a set of benchmark problems, and to focus attention on more realistic problems. Although not intended, the adoption of PDDL as a common representation language for planning was a very important outcome of the first competition. Overall the benefits of this series of competitions have been significant: over ten years, planning systems have been developed that are capable of solving large and complex problems, using richly expressive domain models, and meeting advanced demands on the structure and quality of solutions. The competition series has inspired many advances in the planning research community as well as an increasingly empirical methodology and a growing interest in the application of planners to real problems.

The Prognostics and Health Management (PHM) community held its first prognostics data challenge as part of the PHM'08 conference (Saxena *et al.*, 2008). The challenge was an initial step in addressing the lack of publicly available run-to-failure sets. Fault progression data sets are typically time consuming and expensive to acquire or may be withheld due to proprietary or competitive reasons. The lack of common data sets is seen as impeding progress in the prognostics community as there is no mechanism for a meaningful, direct comparison of algorithms. The PHM data challenge required competitors to estimate the remaining useful life (RUL) of an unspecified system using historical data only. For this competition no framework was required, since the evaluation was with respect to one parameter representing the prognostic output, which did not depend on timing, the CPU type, memory usage, or other measures making the comparison of algorithms more straightforward.

In contrast, the diagnosis community has suffered a distinct lack of similar benchmarks, and it has been difficult to perform a comparative analysis of inference algorithms or model representations. Several researchers have attempted to demonstrate benchmarking capability on different systems. Among these, (Orsagh et al., 2002) provided a set of 14 metrics to measure the performance and effectiveness of prognostics and health management algorithms for US Navy applications (Roemer et al., 2005). (Bartys et al., 2006) presented a benchmarking study for actuator fault detection and identification (FDI). This study, developed by the DAMAD-

[1] http://dx-competition.org/
[2] http://www.isy.liu.se/dx09/

DAMADICS Research Training Network, introduced a set of 18 performance indices used for benchmarking FDI algorithms on an industrial valve-actuator system. 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. Finally, (Simon, et al., 2008) introduced a benchmarking technique for gas path diagnosis methods to assess the performance of engine health management technologies.

The framework presented here adopts some of its metrics from the literature (SAE, 2007; Orsagh et al., 2002; Bartys et al., 2006) and extends prior work in this area by 1) defining a number of benchmarking indices, 2) providing a generic, application independent architecture that can be used for benchmarking different monitoring and diagnosis algorithms, and 3) facilitating the use of real process data on large-scale, complex engineering systems. Moreover, it is not restricted to a single fault assumption and enables the calculation of benchmarking metrics for systems in which each fault scenario may contain multiple faults.

3. DIAGNOSIS ALGORITHM EVALUATION FRAMEWORK

For our proposed diagnosis algorithm evaluation framework we first defined formats for the system catalog, sensor data message, and the diagnosis result (described in detail later in this section). The process of setting up the framework for a selected system is as follows:

- The system is formally 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.
- The set of sensor points is decided and sample data for nominal and fault scenarios are generated.
- DA developers use the system catalog and sample data to create their algorithms with an appropriate interface to the run-time architecture (described later in this section) to receive sensor data and send the diagnosis results.
- A set of test scenarios (nominal and faulty) is selected to evaluate the DA.
- The run-time architecture is used to run the DA on the selected test scenarios in a controlled experiment setting, and the diagnosis results are archived.
- Selected metrics are computed by comparing actual scenarios and diagnosis results from DAs. The metrics can then be used to compute secondary metrics (a ranking score if a competition is being conducted for example)

In the following subsections we describe the constituent pieces of our framework in more detail.

3.1 System Catalog

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 (for examples cf. (Feldman, Provan, & van Gemund, 2007) and the references therein), there is no widely agreed way to represent models and systems. Furthermore, DXC is not a pure algorithmic competition, a DA includes its own system representation (for example a Bayesian net or a system of Qualitative Differential Equations). 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. This is done in the XML system catalog. The rest of the system description (e.g., nominal and faulty functionality of components) is not regulated by DXC, i.e., the organizers may provide any textual, programmatic or other description of the systems. 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 catalog format is primarily intended to provide a common set of identifiers for components and their modes of operation within a given system. This common language is necessary to enable exchange of meaningful 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 prevent biasing towards any one diagnostic approach.

Table 1: Top-level system description

Item	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

Almost any diagnosis technology today uses some kind of graph-like structure for describing the system structure. Hence we chose a graph-like representation to specify the physical connectivity of the system. This graph is not annotated: for example there is no directional information. Supplemental information can be extracted from the repository of documents describing the system, if available.

This is an example XML system description:

```
<system xsi:type="Digital Circuit">
  <systemName>polycell</systemName>
  <description>
    A familiar circuit. Contact: dekleer@parc.com.
    Publications: [dW87]
  </description>

  <components>
     <component xsi:type="Probe">
        <name>A</name>
        <description>
          Probes a point in circuit
        </description>
     </component>
     <component xsi:type="Multiplier">
        <name>M1</name>
        <description>
          Multiplies its inputs
        </description>
     </component>
     <component xsi:type="Adder">
        <name>A1</name>
        <description>Adds its inputs</description>
     </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, specifications for all component types mentioned in the system description are provided.

Table 2: Component description data

Item	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
Component Specific Info	Examples: sensor min/max, load wattage, circuit breaker rating

Consider the "Circuit Breaker" component type (referenced, for example, by a component with unique ID CB180):

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

Or an "AC Voltage Sensor" component type:

```
<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>
```

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

```
<componentType xsi:type="ANDGate">
  <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 3. The mode catalog organizes nominal and faulty behavior of under generic component types. As an example, the allowable modes for the Circuit Breaker component from the preceding section are given below:

Table 3: Mode group description

Item	Description
Modes Group Name	Unique identifier for each mode group
Mode Names	Names of the possible modes
Mode Descriptions	Text descriptions

```
<modeGroup>
  <name>CircuitBreaker</name>
  <mode xsi:type="mode">
    <name>Nominal</name>
    <description>
      Transmits current and voltage.
      Trips when current exceeds threshold.
    </description>
  </mode>
  <mode xsi:type="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>
    <faultSource>Hardware</faultSource>
    <parameters/>
  </mode>
</modeGroup>
```

The mode catalog and the specific mode definitions serve as a requirements document for diagnosis algorithm developers. This is intended to establish a common ground for different approaches by guiding the modeling of physical systems to use a standardized nomenclature.

3.2 Message Format

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 benchmarking are the sensor data message, command message, and diagnosis message, described below.

3.2.1 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.

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 value).

Table 4: Sensor and command messages

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

CommandMessage
+timestamp
+commandID: string
+command: commandValue

3.2.2 Diagnosis Result Format

The diagnosis 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 benchmarking metrics, which will be introduced in Section 3.5. The resulting diagnostic message is summarized in Table 5 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.

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.

Table 5: The diagnosis message

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

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

3.3 Run-time Architecture

The key component of our framework is the run-time architecture for executing and evaluating diagnosis algorithms. The architecture has been designed with the following considerations in mind:

1. The overhead of interfacing existing diagnosis algorithms should be reduced by supplying minimalistic APIs
2. Inter-platform portability should be provided by allowing clients to interface C++ and Java APIs or by implementing a simple ASCII based TCP messaging protocol

To facilitate algorithm development and testing we intend to make available all software in source form and binary packages for Windows™ and Linux platforms. Our framework will contain a very simple diagnosis algorithm and a few examples.

3.3.1 Software Components

Figure 1 shows an overview of the software components and the primary information flow the framework. All communication is ASCII based and all modules communicate via TCP ports by using a simple message-based protocol which we will describe in more detail below.

- **Scenario Loader** (SL): This is the main entry point for running the diagnosis algorithms. The Scenario Loader (SL) executes the Scenario Data Source (SDS), the Scenario Recorder (SR), and all Diagnosis Algorithms (DA).

SL ensures system stability and cleanup upon scenario completion. Note that SL is the only long living process. SDS, SR, and all DAs are spawned for each scenario and the DA is forcibly killed if it does not terminate after a predetermined time-out.

- **Scenario Data Source** (SDS): SDS 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 discern 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, and compiles it into a Scenario Results File. The results file contains a number of time-series which will be described later. 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.

- **Diagnosis Algorithm** (DA): This is the component implemented by the external participants for evaluation. DA implementations use the diagnostic framework messaging interface to receive sensor and command data, perform their diagnosis, and send the results back.

- **Evaluator**: Takes Scenario Results File and applies metrics to evaluate Diagnosis Algorithm performance. The metrics and evaluation procedures are detailed in Section 6.

3.3.2 Diagnostic Session Overview

The diagnosis algorithms are tested against a number of diagnostic scenarios. From the viewpoint of the scenario player, a diagnostic scenario is a series of observations (sensor readings, commands) $\mathbf{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, taken within an interval of time. $\mathbf{A}$ is computed for each scenario with the help of a physical testbed, a simulator, or other methods for creating observations (it is not necessary for all available sensor values to be reported to the diagnosis engine). The aim should be to provide scenarios with varying levels of difficulty. The diagnostic scenarios are kept secret from the participants, except those provided as training and test data.

We will analyze the progression of one diagnostic scenario. Each diagnostic session defines some standard key points and intervals which are best illustrated by Figure 2.

Figure 2 splits the diagnostic session into three important time intervals: $\Delta_{startup}$, $\Delta_{injection}$, and $\Delta_{shutdown}$. During the first interval $\Delta_{startup}$, the diagnosis algorithm is given time to initialize, read data files, etc. Note that this is not the time for compilation; compilation-based algorithms will compile their models beforehand. Though sensor observations may be available during $\Delta_{startup}$, no faults will be injected during this time. Fault injection will take place during $\Delta_{injection}$. Finally, the algorithms will be given some time post-scenario to send final diagnoses and gracefully terminate during $\Delta_{shutdown}$. After this time, live diagnosis processes will be killed and the system will be recycled for the next diagnostic experiment.

Below are some notable points for the example diagnostic scenario from Figure 2:

- t_{inj} – A fault is injected at this time;
- t_{fd} – The diagnosis algorithm has detected a fault;
- t_{ffi} – The diagnosis algorithm has isolated a fault for the first time;
- t_{fir} – The diagnosis algorithm has modified its isolation assumption;
- t_{lfi} – This is the last fault isolation during $\Delta_{injection}$.

A sequence diagram of an example diagnostic session is shown in Figure 3. After Scenario Loader spawns the Scenario Data Source, Scenario Recorder, and Diagnostic Algorithm, the three spawned processes send messages indicating they are prepared to start the scenario. Scenario Loader notifies Scenario Data Source that it should commence sending sensor data, command data, and fault injection information. Scenario Recorder receives all three kinds of data; the DA only sensor and command data. The DA in turn sends its diagnoses to the Scenario Recorder.

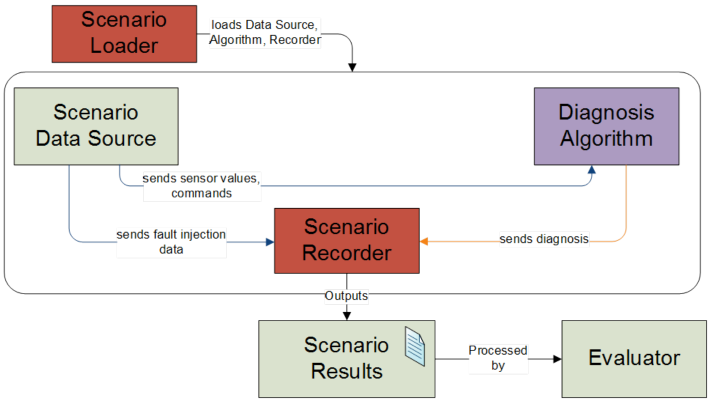

Fig. 1. Run-time architecture

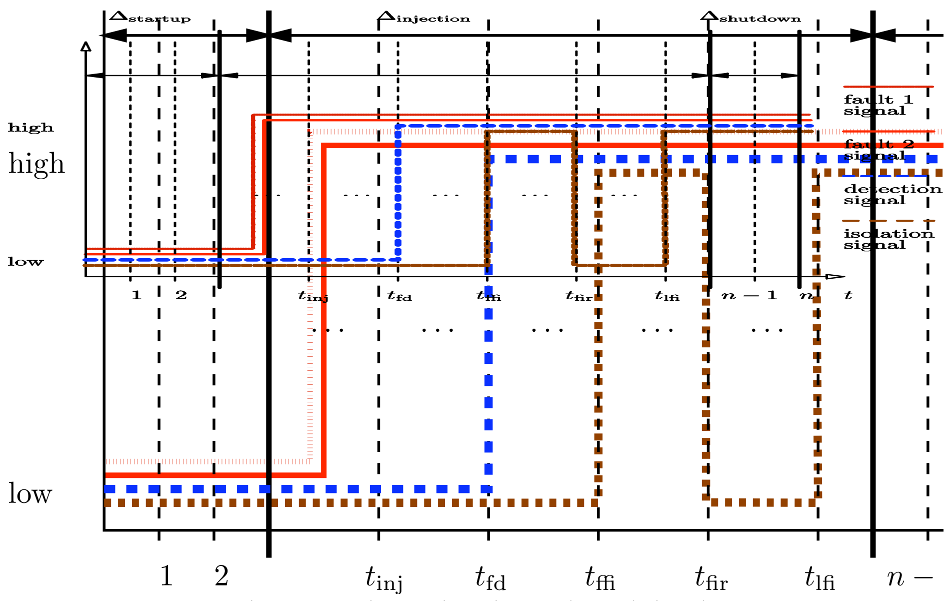

Fig. 2. Key time points, intervals, and signals

At the end of the diagnostic session the scenario player has collected the following time-series and (actual and hypothesized) fault data to be used in the metrics computation:

- Fault injection signal; (from Scenario Data Source)
- Fault detection signal; (from DA)
- An actual fault set (once all faults are injected); (from Scenario Data Source)
- A (possibly empty) set of candidate diagnoses; (from DA)

In addition, the scenario loader collects the following diagnosis engine session performance data (to be used for the computation of performance metrics):

- Total computation time; (from Operating System)
- Peak amount of allocated memory; (from Operating System)

3.4 Metrics and Evaluation

The metrics for evaluating diagnostic algorithms depend on the particular use of the diagnostic system, the users involved, and their objectives. In this paper, we have chosen to highlight some common requirements for the development of diagnostic systems. The metrics measuring the performance on these requirements need to be defined separately depending on the specific diagnostic application at hand. For the proposed framework, we make a distinction between *temporal*, *technical*, and *computational* performance and highlight metrics for each category.

The *temporal metrics* measure how quickly an algorithm responds to faults in a physical system. The *technical metrics* measure non-temporal features of a diagnostic algorithm including accuracy, resolution, sensitivity, and stability. Finally*, computational metrics* are intended to measure how efficiently an algorithm uses the available computational resources. An extensive survey on these user requirements and associated metrics can be found in Kurtoglu et al. (2008).

In addition we divide the metrics into 2 main categories:

- *Detection metrics* which deal with temporal, technical and computational metrics associated with only detection of the fault.
- *Isolation metrics* which deal with temporal, technical and computational metrics associated with isolation of the fault.

Table 6: Metrics summary

Name	Description	Class/Category
"Per System Description" Metrics		
False Positives Rate	Spurious faults rate	Technical / Detection
False Negatives Rate	Missed faults rate	Technical / Detection
Detection Accuracy	Correctness of the detection	Technical / Detection
"Per Scenario" Metrics		
Fault Detection Time	Time for detecting a fault	Temporal / Detection
Fault Isolation Time	Time for last persistent diagnosis	Temporal / Isolation
Fault Isolation Accuracy	Correctness of Isolation	Technical / Isolation
CPU Load	CPU time spent	Computational / Detection & Isolation
Memory Load	Memory allocated	Computational / Detection & Isolation

In general several other classes of metrics are possible including cost/utility metrics, effort (in building systems for example) metrics and also other categories like fault identification and fault recovery metrics. The expectation is that as this framework 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.

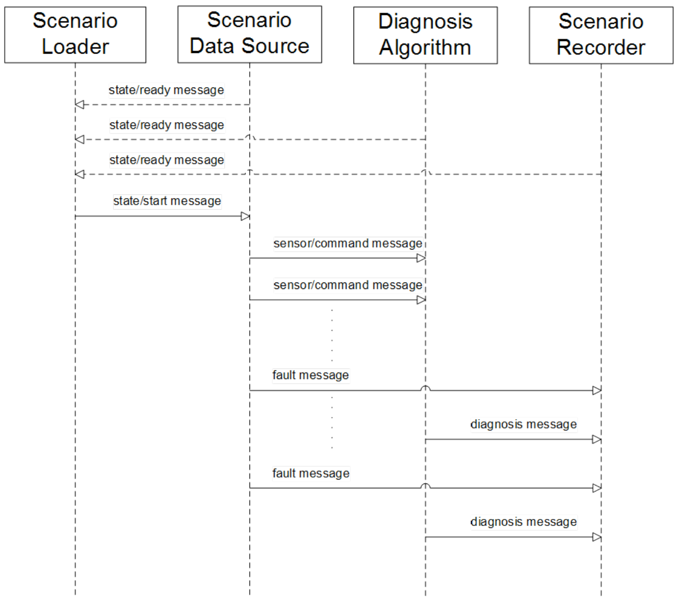

Fig. 3: Session sequence diagram.

For the first implementation of the evaluator (Figure 1) we included 8 metrics which are summarized in Table 6. These metrics are based on extensive survey of literature and talking to experts from various fields (Kurtoglu et al., 2008). The definitions for the eight metrics are provided in (Kurtoglu et al., 2009).

4. ASSUMPTIONS, ISSUES, AND EXTENSIONS

In developing this framework, we made a few simplifying assumptions and identified some limitations, which form the scope for future extensions.

The system catalog was 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. We expect to be continuously updating the diagnosis result format to support a variety of diagnostic output.

The run-time architecture was also defined such that no assumptions were made regarding the actual operational environments in which the diagnostic algorithms may be run. We acknowledge that the true test of diagnostics is in robustness of performance in the target environment, however, building an effective evaluation framework in an operational environment is prohibitive for almost all application domains.

The set of metrics we chose are 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 importance (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).

5. CONCLUSIONS & FUTURE WORK

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 diagnosis algorithm and the framework includes tools to compute the metrics by comparing actual scenarios and diagnosis results.

We identified some of the limitations of this framework in Section 4. Our goal is to continue extending this framework to address those limitations. One of our immediate future goals is to augment the presented framework with Internet accessible data files that would enable testing of various approaches and strategies in a web-based, distributed fashion. In addition, we plan to extend the framework to make it compatible and interoperable with industry standard representations and diagnostic information models including AI-ESTATE (Sheppard and Kaufman, 1999) and Hybrid System Interchange Format[3] (HSIF).

[3] http://w3.isis.vanderbilt.edu/Projects/mobies/downloads.asp#HSIF

Finally, in a related effort, we used this framework to implement a Diagnosis Competition (DXC'09) (Kurtoglu et al., 2009). The Diagnostic Competition is the first of a series of international competitions that will be hosted at the International Workshop on Principles of Diagnosis (DX). The overall goal of the competition is to systematically evaluate different diagnostic technologies and to produce comparable performance assessments of different diagnostic methods.

ACKNOWLEDGMENTS

We extend our gratitude to Peter Struss (Technical University Munich), Gautam Biswas (Vanderbilt University), Ole Mengshoel (Carnegie Mellon University), Serdar Uckun (PARC), Kai Goebel (University Space Research Association), Gregory Provan (University College Cork), and many others for valuable discussions, criticism and help.

REFERENCES

Bartys, M., R. Patton., M. Syfert, S. de las Heras, and J. Quevedo (2006). Introduction to the DAMADICS Actuator FDI Benchmark Study, *Control Engineering Practice*, 14, pp.577-596.

Basseville M. and I. Nikiforov (1993). Detection of Abrupt Changes, Prentice-Hall, Inc., Englewood Cliffs, NJ.

de Freitas N. (2001). Rao-Blackwellised Particle Filtering for Fault Diagnosis. In: *IEEE Aerospace Conference*.

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

Gertler, J. and NetLibrary Inc. (1998). Fault detection and Diagnosis in Engineering Systems. New York: Marcel Dekker.

Hamscher, W., L. Console and J. de Kleer (1992). Readings in model-based diagnosis. *Morgan Kaufmann,* San Mateo, CA.

Iverson, D., Inductive System Health Monitoring (2004). In: *Proc. ICAI'04*.

Kavcic, M. and D. Juricic (1997). A prototyping tool for fault tree based process diagnosis. In: *Proc. DX'97*.

Kostelezky, W., et al. (1990). The Rule-based Expert System Promotex I. Sttutgart. *ESPRIT Project #1106*.

Kurtoglu T., O. J. Mengshoel, and S. Poll (2008). A Framework for Systematic Benchmarking of Monitoring and Diagnostic Systems, In: *Proc. PHM'08*.

Kurtoglu T., S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, J. de Kleer, A. van Gemund, A. Feldman (2009). First International Diagnosis Competition – DXC'09. In: *Proc. DX'09*.

Lerner, U., R. Parr, D. Koleer, G. Biswas (2000). Bayesian Fault Detection and Diagnosis in Dynamic Systems. In: *Proc. AAAI'00*, pp. 531-537.

Orsagh R., Roemer, M., Savage, C., and Lebold, M., (2002). Development of Performance and Effectiveness Metrics for Gas Turbine Diagnostic Techniques. *Aerospace 2002 IEEE Conference Proceedings*, 6, pp. 2825-2834.

Poll, S., A. Patterson-Hine, J. Camisa, D. Garcia, D. Hall, C. Lee, O. J. Mengshoel, C. Neukom, D. Nishikawa, J. Ossenfort, A. Sweet, S. Yentus, I. Roychoudhury, M. Daigle, G. Biswas, and X. Koutsoukos (2007). Advanced Diagnostics and Prognostics Testbed. In: *Proc. DX'07*.

Roemer, M, J. Dzakowic, R. Orsagh, C. Byington, and G. Vachtsevanos (2005). Validation and Verification of Prognostic Health Management Technologies. In: *Proc. AEROCONF'05*.

Roozbeh I., M. Blanke, A ship propulsion system as a benchmark for fault-tolerant control, *Control Engineering Practice,* 7(2), pp. 227-239.

SAE (Society of Automotive Engineers) E-32, 2007, Health and Usage Monitoring Metrics, Monitoring the Monitor, February 14, 2007, *SAE ARP 5783-DRAFT*.

Saxena, A., K. Goebel, D. Simon, and N. Eklund (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. In: *Proc. PHM'08*.

Sheppard, J. and M. Kaufman (1999), IEEE Information Modeling Standards for Test and Diagnosis. In: *Proc. 5-th Annual Joint Aerospace Weapon System Support, Sensors and Simulation Symposium*.

Simon L., J. Bird, C. Davison, A. Volponi, R. E. Iverson, (2008). Benchmarking Gas Path Diagnostic Methods: A Public Approach, *Proceedings of the ASME Turbo Expo 2008: Power for Land, Sea and Air, GT'08*.

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