

Standard Representations of Diagnostic Models

W. R. Simpson, IDA
IDA
1801 N. Beauregard St.
Alexandria, VA 22311
rsimpson@ida.org

J. W. Sheppard
ARINC
2551 Riva Road
Annapolis, MD 21401
jsheppard@arinc.com

ABSTRACT

We present a description of the AI-ESTATE (IEEE 1232.1) standard for representing and exchanging diagnostic models. These models are based on accepted approaches to performing system diagnostics in both commercial and military environments. Specifically, we will discuss a standard approach to representing diagnostic fault trees and enhanced diagnostic inference models.

1. INTRODUCTION

Many approaches exist for performing fault diagnosis including methods for improving the diagnostic process have been studied, with particular attention coming from the artificial intelligence community. A number of tools have been developed that use modeling of one form or another to provide a structure that allows diagnostic reasoning to be undertaken for both testability assessment and diagnostic strategy development. One formal approach from artificial intelligence that facilitates developing diagnostics is derived from the principles of formal logic. This approach uses enhanced diagnostic inference models (EDIMs) to capture logical relationships between tests and faults in a unit. These models have been derived from several commercially available tools, including STAT, STAMP, TEAMS, WSTA, and AI-TEST¹⁻⁵.

Recent initiatives by the Institute of Electrical and Electronics Engineers (IEEE) on standardizing test architectures have provided a unique opportunity to improve the development of test systems. The IEEE P1232 "Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE)⁶" initiative is attempting to usher in the next generation in product diagnostics by standardizing diagnostic services and development tool interfaces. By using problem encapsulation, defining interface boundaries, developing exchange formats and specifying standard services, AI-ESTATE provides a methodology for developing diagnostic systems that will be interoperable, have transportable software, and move beyond vendor and product specific solutions.

The concepts in the AI-ESTATE standard are not limited to the specific area of test, but apply to manual, automatic, and semi-automatic test, as well as the domains of electronic, mechanical, pneumatic, and other types of systems. The AI-ESTATE subcommittee designed the P1232 standards to abstract specific test and product details out of the diagnostic models and to tie these models to domain-specific models as needed to complete the test system. This paper is necessarily

limited in scope. The paper provides an overview of the diagnostic models as provided by the AI-ESTATE standards⁷.

2. AI-ESTATE

The AI-ESTATE subcommittee has established several ambitious goals for the AI-ESTATE standards that include:

- Provide a standard interface between diagnostic reasoners and other functional elements that reside within an AI-ESTATE system.
- Provide formal data specifications to support the exchange of information relevant to the techniques commonly used in system test and diagnosis.
- Maximize compatibility of diagnostic reasoning system implementations.
- Accommodate embedded, coupled, and stand-alone diagnostic systems.
- Facilitate portability, reuse, and sharing of diagnostic knowledge.

To achieve these goals, the AI-ESTATE subcommittee proceeded to define an architecture for a standard diagnostic system and then defined component standards for information exchange and software interfaces.

An Architecture for Diagnosis

The AI-ESTATE architecture presented in Figure 1 shows a conceptual view of an AI-ESTATE-conformant system⁶. AI-ESTATE applications may use any combination of functional elements and interfunction communication as shown in the figure. The service specification (P1232.2)⁸, or other specifications relevant to the particular functional element, define the form and method of communication between reasoning systems and other functional elements. AI-ESTATE identifies reasoning services provided by a diagnostic reasoner so that transactions between test system components and the reasoner are portable.

AI-ESTATE assumes a client-server or cooperative processing model in defining the diagnostic services. One significant difference between a traditional client-server model and AI-ESTATE is the assumption of a possibly abstract application executive mediating service requests. In some sense, this application executive can be regarded as a service broker for the subsystems in the test environment. Services are "published" to the application executive, and service requests from other subsystems are matched to the available services to satisfy the request. This idea is analogous to the CORBA

architecture except with no underlying assumption of being object-oriented. Further, since the application executive can be abstract, it is still possible for subsystems to interact directly with other subsystems using their published services.

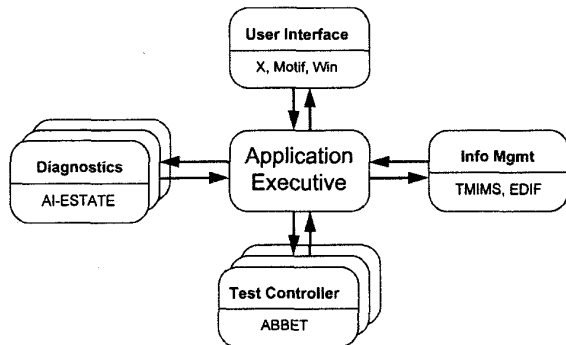


Figure 1. AI-ESTATE Architecture

-From the vantage point of an AI-ESTATE diagnostic reasoner, one sees the interaction with the application executive on two planes. First, the AI-ESTATE reasoner makes available several services (as defined by IEEE P1232.2) to the application executive for traversing diagnostic models or actually performing diagnosis given test results. Second, the diagnostic reasoner interfaces with other subsystems in the test environment (e.g., the test system) by requesting services from the application executive. For example, while the reasoner will not perform any tests, it is likely to request certain tests be performed in certain contexts. The application executive will be used to submit the request to the test system.

In addition to interfacing with the application executive, it is assumed the AI-ESTATE reasoner has direct access to diagnostic models. IEEE Std 1232.1 provides a means for exchanging models between conformant reasoners, and this exchange can either be accomplished using model traversal services or using the interchange format defined in 1232.1.

Diagnostic Models

The current version of IEEE Std 1232.1 defines three models for use in diagnostic systems—a common element model, a fault tree model, and an enhanced diagnostic inference model. All of the models were defined using ISO 10303-11, EXPRESS⁹. EXPRESS is a language for defining information models and has received widespread acceptance in the international standards communities of ISO and IEC. For example, EDIF 3 0 0 and EDIF 4 0 0 were defined using EXPRESS.

The common element model defines information entities, such as a test, a diagnosis, an anomaly, and a resource, which are expected to be needed by any diagnostic system. The common element model also includes a formal specification of costs to be considered in the test process. The remaining two models represent knowledge that may be used by specific types of diagnostic systems. The fault tree model defines a decision tree based on outcomes from tests performed by the test system. Each node of the tree corresponds to a test with some set of outcomes. The outcomes of the tests are branches extending

from the test node to other tests or to diagnostic conclusions (such as *No Fault*). Typically, test programs are designed around static fault trees; therefore, the AI-ESTATE subcommittee decided to include a representation for a fault tree in the standard, even though fault trees are not typically considered to be AI systems.

The AI-ESTATE fault tree model imports elements and attributes from the common element model. Typically, test systems process fault trees by starting at the first test step, performing the indicated test, and traversing the branch corresponding to the test's outcome. The test program follows this procedure recursively until it reaches a leaf in the tree, indicating it can make a diagnosis.

The enhanced diagnostic inference model (EDIM) is based on the dependency model. Historically, test engineers used the dependency model to map relationships between functional entities in a system under test and tests that determine whether or not these functions are being performed correctly. In the past, the model characterized the connectivity of the system under test from a functional perspective using observation points (or test points) as the junctions joining the functional entities together. If a portion of the system fed a test point, then the model assumed that the test associated with that test point *depended* on the function defined by that part of the system.

Recently, researchers and practitioners of diagnostic modeling found that the functional dependency approach to modeling was problematic and could lead to inaccurate models. Believing the algorithms processing the models were correct, researchers began to identify the problems with the modeling approach and to determine how to capitalize on the power of the algorithms without inventing a new approach to model-based diagnosis. They found that the focus of the model should be on the tests and the faults those tests detect rather than on functions of the system. In particular, the focus of the model shifted to the inferences drawable from real tests and their outcomes, resulting in a new kind of model called the "diagnostic inference model." The *enhanced* diagnostic inference model, defined by AI-ESTATE, generalizes the diagnostic inference model by capturing hierarchical relationships and general logical relationships between tests and diagnoses.

The information models defined in the AI-ESTATE standard, by themselves, provide a common way of talking about the information used in diagnosis, but this is not enough for a standard. In AI-ESTATE, these models also provide the basis for a neutral exchange format. Using this neutral format, multiple vendors can produce diagnostic models in the format to enable their use by other tools that understand that format.

To specify the neutral exchange format, the AI-ESTATE subcommittee decided to use an instance language defined by the ISO STEP (Standards for the Exchange of Product data) community based on EXPRESS—EXPRESS-I¹⁰. EXPRESS-I is an instance language defined to facilitate developing example instances of information models and to facilitate developing test cases for these models.

As an alternative, the ISO STEP community has defined a standard physical file format derived from EXPRESS models. Unfortunately, the STEP physical file format is very difficult for a human to read but very easy for a computer to process.

The AI-ESTATE subcommittee found added benefit in EXPRESS-I over the STEP physical file format in that the language is both computer-processable and human-readable.

Diagnostic Services

The AI-ESTATE standard defines several software services to be provided by a diagnostic reasoner. The nature of these services enables the reasoner to be embedded in a larger test system; however, it is possible that the diagnostic system is a stand-alone application connected to a graphical user interface of some kind. Currently, the services defined by AI-ESTATE are classified as either static model accessor services, or reasoner state accessor services. Following the object-oriented programming paradigm, we found that all services could be represented in one of four forms: create, get, put, or delete.

3. AI-ESTATE DATA AND KNOWLEDGE SPECIFICATION

Currently, IEEE Std 1232.1-1997 defines three information models to be used for knowledge exchange:

- The common element model
- The fault tree model
- The enhanced diagnostic inference model

AI-ESTATE does not anticipate these being the only models included in the standard, but they provide both a baseline model (the fault tree model) and a widely used, successful model for dynamic diagnosis (the EDIM). In addition, the common element model has been designed to cover the basic entities expected to be used by any reasoning model. Future efforts in model are expected to cover constraint models, neural network models, belief network models, and first-order rule-based models.

Common Element Model

The common element model defines a top-level structure to be used by all specific reasoning models. Since the AI-ESTATE philosophy emphasizes abstraction and separation of test/fault details from diagnostics, the entities in the common element model do not contain sufficient information for testing. The CEM does provide an arbitrary structure for organizing tests, diagnoses, anomalies, and resources. This arbitrary structure is defined in terms of a lattice in which it is assumed top-down relationships exist and no cycles exist in the models. In addition to defining the primary entities used for diagnosis, the CEM provides a flexible cost model to be used by a diagnostic reasoner in optimization.

The simplified view of the CEM (Figure 2) shows, explicitly, the separation between test and diagnosis. Diagnoses correspond to the conclusions to be drawn by the reasoner, and the anomalies correspond to "physical" conditions in the unit being tested. Thus a diagnosis (conclusion) is mapped to an anomaly (physical condition). Tests require resources to be executed and generate outcomes to be used in diagnosis. The attribute for resource is given as an optional attribute since resources may be handled external to the reasoner. Even if resources are specified, the resource model within AI-ESTATE

is not sufficiently robust to capture all of the needed information about the resource. AI-ESTATE expects the TRIM to provide the needed information. Note no relationships between tests and diagnoses are shown in this model. Such relationships are the purpose of the reasoning models which provide the *logical* relationship between test and diagnosis.

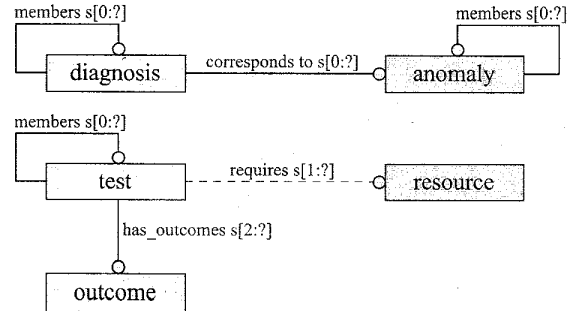


Figure 2. Simplified Common Element Model

AI-ESTATE assumes the potential for multiple models to be available for diagnosis. All of these models are "wrapped up" into a container called diagnostic knowledge which defines the knowledge base for this unit and context. For purposes of optimization, cost is only associated with tests and resources. Note the cost elements specified in AI-ESTATE are intended to provide expected costs rather than actual costs.

In addition, outcomes have confidences associated with them. The confidences are associated with the outcomes rather than the tests since this is more general. It is expected that these confidences do not define "actual" confidences from testing but "expected" confidences.

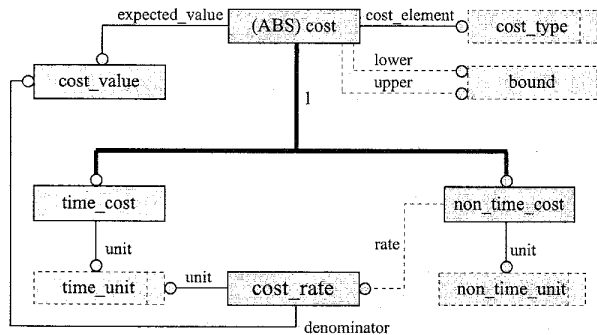


Figure 3. Simplified Cost Model

The simplified view of the cost model (Figure 3) illustrates the distinction between time cost and non-time cost. Both cost entities are intended to be "abstract" in the sense that any cost measure can be used (subject to the types of units specified). While many diagnostic systems may not make explicit use of cost information, a cost model was necessary to support maintenance feedback and dynamic diagnosis in which there is an attempt to optimize the process. For each cost entity, a specific cost "type" can be assigned. These types correspond to setup cost, performance cost, re-entry cost, and access. In addition, a "user-defined" type is permitted.

The primary entity of the cost model is `cost`. This entity, is an abstract supertype and cannot be instantiated by itself. Rather, it provides a higher order entity with common attributes to be inherited by the subtypes. Two subtypes are defined: `time_cost` and `non_time_cost`. The only real difference between these two types of costs is the set of units available. In addition, the `non_time_cost` entity can be specified in terms of some cost per unit time.

Since diagnostics frequently focuses on probabilistic measures, it was necessary to form some common basis for computing these measures. Generally, such measures are based on reliability statistics, the most commonly used statistic being failure rate. For consistency, it was decided that all failure rates would be provided per million hours. This can be converted easily to any other basis if needed.

Fault Tree Model

The first "reasoner" model developed for AI-ESTATE was the fault tree model. While not generally considered to be used in artificial intelligence, this model provides a baseline for AI-ESTATE since most test environments in existence today follow some kind of fixed test strategy that can be modeled as a fault tree. Even so, the AI community generally regards "decision trees" to be special forms of rule bases. Since a fault tree is a decision tree, fault trees also form the basis for a restricted type of rule base.

The AI-ESTATE fault tree also extends the notion of the traditional fault tree in two ways. First it permits test confidences to be processed to establish a level of "belief" in the reported diagnosis. Second, it provides the capability of reporting "intermediate diagnoses" in the interior of the tree rather than waiting until the leaves of the tree to report diagnostic information.

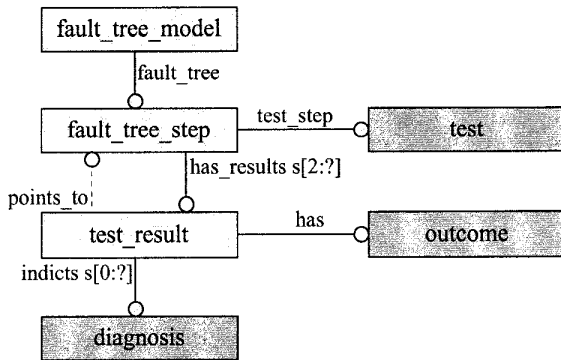


Figure 4. Simplified Fault Tree Model

The core of the fault tree model (Figure 4) references three entities (shown in gray) from the common element model—`test`, `diagnosis`, and `outcome`. The fault tree provides an arbitrary number of outcomes to be processed (due to the arbitrary outcomes that can be associated with a test in the common element model). Thus an AI-ESTATE fault tree need not be restricted to a binary decision tree. The only attribute of the `fault_tree_model` entity is a pointer to the first step of the tree. Each step has a single test associated with it and a set

of results. Each of these results corresponds to each of the legal outcomes of that test. The test result then points to either the next step in the tree or to a diagnosis.

The fault tree model is a subtype of the common element model's `diagnostic_model`. This construct indicates a problem in EXPRESS in that EXPRESS does not handle subtyping across schemata well. Nevertheless, for purposes of the exchange format and the data typing, we are still able to use the EXPRESS construct to show the needed relationship.

The following example shows a small portion of a fault tree specified in EXPRESS-I. Note the fault tree model assumes the incorporation of entities from the common element model. These entities are not shown in the example. In this example, the first entity, given as `FT`, identifies the fault tree and points to the first step (`Step1`) of the tree.

```

FT = fault_tree_model{
    SubOf(@model1);
    fault_tree -> @Step1;
};

Step1 = fault_tree_step{
    test_step -> @t3;
    result ->
        (@S1_Result1,@S1_Result2);
};

S1_Result1 = test_result{
    test_outcome -> @t3_pass;
    next_step -> @Step2;
    current_diagnosis -> ();
};
  
```

One distinction between EXPRESS and EXPRESS-I is that EXPRESS does not require supertype relationships to be explicitly identified in the model; however EXPRESS-I does. Since the common element model entities are explicitly incorporated into the instantiated model, this is not a problem. Instantiating the knowledge base will include all instantiated diagnostic models as well, and the supertype/subtype relationships can be specified explicitly.

`Step1` shows we are considering test `t3` which has two results, `S1_Result1` and `S1_Result2` corresponding to the test passing or failing respectively. Assuming the test passes, we go to `S1_Result1` which points to the next step in the tree, `Step2`. In this example, no intermediate diagnosis is provided.

Enhanced Diagnostic Inference Model

The "primary" reasoning model defined in AI-ESTATE is the enhanced diagnostic inference model (EDIM). Philosophically, the EDIM is derived from the assumption that information provided by tests is what matters rather than focusing on the diagnoses that might be drawn. The EDIM does not explicitly include sequence information but instead records logical relationships between tests and diagnoses.

Since sequencing can be a significant concern, AI-ESTATE provides a means for coupling EDIMs and fault trees together

by way of the diagnostic_knowledge entity in the common element model. At any point, if a test is selected that is simultaneously a test in the EDIM and an initial test in a fault tree, the application executive can shift from the EDIM to the fault tree and follow the predefined sequence. When the fault tree terminates, if tests remain that could further resolve the diagnoses, the application executive can shift back to the EDIM to continue.

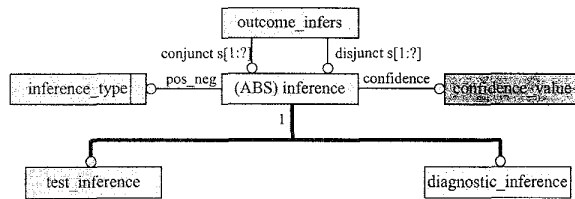


Figure 5. Simplified Enhanced Diagnostic Inference Model

As shown in Figure 5, the core of the EDIM is the entity, `outcome_infers`. This entity is associated with a particular test outcome in the common element model. Recall that each test outcome is unique and tied to a specific test. An EDIM is nothing more than a collection of `outcome_infers` instances. Associated with each of these entities is a list of conjuncts (AND'ed inferences) and disjuncts (OR'ed inferences).

Associated with each inference is confidence in that inference being drawn and an indication of whether or not the inference is positive or negative. A positive inference is simply an inference made. A negative inference is the NOT of the indicated inference. The inference entity is actually an abstract supertype of either a test inference or a diagnostic inference. Thus, with any test, the state of a test (i.e., its outcome) or diagnosis (i.e., whether or not it remains a candidate) can be inferred.

As with the fault tree model, the EDIM is a subtype of the entity `diagnostic_model` from the common element model (Figure 16). The EDIM must contain at least two `outcome_inference` entities; otherwise, the inference is trivial and a model is not needed.

The structure of the EDIM is very simple and, somewhat surprisingly, very general. In fact, given this structure, any arbitrary logical expression can be constructed and tied to a test outcome. Thus the EDIM provides a significant enhancement over the traditional DIM that is limited to binary outcomes, conjuncted inferences on passing tests, and disjuncted inferences on failing tests. (Actually, many implementations of DIMs are slightly more general than this.)

The following example shows a small portion of an EDIM derived from a digital fault dictionary. As described earlier, the EDIM is really nothing more than a set of inferences as shown in the `inference` attribute of `edim`.

```

edim =
  enhanced_diagnostic_inference_model{
    SubOf(@model1);
    inference ->
      (@t1_pass_implies,@t1_fail_implies,

```

```

      @t2_pass_implies,@t2_fail_implies,
      @t3_pass_implies,@t3_fail_implies,
      @t4_pass_implies,@t4_fail_implies);
  };

t1_pass_implies = outcome_inference{
  test_outcome -> @t1_pass;
  conjuncts ->
    (@x_sal_absent,@y_sal_absent,
     @S_sal_absent,@C_sal_absent);
  disjuncts -> ();
};

x_sal_absent = inference{
  SupOf(@x_sal_elim);
  pos_neg ->
    inference_type(!positive);
  confidence ->
    confidence_value(0.99);
};

```

For a particular inference (`t1_pass_implies`), we see the identification of the actual test outcome (`t1_pass`) and the inferences that can be drawn. Note the test outcome points to the respective test as one of its attributes thus eliminating the need for the inference to point directly to the test. In this example, the inferences are limited to a set of AND'ed diagnostic inferences in which several candidate faults are eliminated from consideration.

Extensibility

A significant issue relevant to any standardization activity is extensibility. Ideally, one would like to be able to readily identify extensions to a standard for incorporation in the next revision of the standard. In addition, one would like to control extensions in such a way that they do not "invalidate" the information specified in the standard.

AI-ESTATE chose to apply an idea from ATLAS to control extensibility. Analogous to the EXTEND construct in ATLAS, AI-ESTATE defines an EXTEND schema. Essentially, AI-ESTATE has no model entity for extensions but permits EXPRESS to be used as is for extending the models. To control the extensions, AI-ESTATE imposes several rules for creating new model entities:

- Legal EXPRESS but no redefinition of existing 1232.1 schemata.
- Must use only types defined in 1232.1 or base.
- Entities must be subtypes of 1232.1 entities.
- Cannot invalidate conclusions drawn from a 1232.1-only model.
- Extensions must be labeled with prefix of `extend_`.
- Use REFERENCE FROM to incorporate 1232.1 schemata.

The essence of the rules for extensibility is that all extensions must be readily identified by examining the information model, all new entities must be subtyped from existing entities, no existing entities can be "redefined", and no inference drawn on the unextended model can be invalidated by the extended model. If these rules end up being too restrictive, then those "extending" in such a way that violate the rules become non-

conformant. It is expected that any use of the non-conformant extensions will identify areas where the AI-ESTATE standards need to be updated prior to release as a "full-status" standard. At that time, it is expected that the AI-ESTATE standard will be specified on the critical interfaces list by the Executive Agent's Office.

In the following example, we identify the schema as an extension by prepending `extend_` to the name of the schema.

```

SCHEMA extend_schema_a;

REFERENCE FROM
    ai_estate_common_element_model;

TYPE extend_type1 = STRING;
END_TYPE;

TYPE extend_type2 = REAL;
END_TYPE;

ENTITY extend_ent_a;
    SUPERTYPE OF (extend_ent_b)
    SUBTYPE OF (outcome);
    att1 : extend_type1;
END_ENTITY;

ENTITY extend_ent_b;
    SUBTYPE OF (extend_ent_a);
    att2 : extend_type2;
    att3 : SET [0:?] OF cost;
END_ENTITY;

END_SCHEMA;

```

Existing entity definitions are incorporated from the common element model. If this was an extension to the fault tree, entity definitions from the fault tree would be incorporated as well. In fact, it is likely that if the common element model is extended, one or more of the reasoning models would need to be extended too.

New types are defined within the EXTEND schema using either base types or previously defined AI-ESTATE types. These new types are used in defining the extended entities. Only the extended entities are included in the EXTEND schema since previously defined entities cannot be redefined. The extended entities, rather than redefining existing entities, subtype existing entities. To control arbitrary extensions, all extended entities must be subtypes of a previously defined entity.

4. SUMMARY

Given the assumptions of the AI-ESTATE architecture, the models and services of AI-ESTATE have broad applicability to advanced diagnostics. As claimed in the acronym, it is believed that AI-ESTATE covers all essential elements of diagnostic reasoners in all test environments. Nothing in the standard

limits the applicability of the standard to a particular approach to test.

The principal assumption of AI-ESTATE is separation of test and diagnosis. Without this separation, the standards cannot be used effectively. By separating the diagnostics from the test process, it also provides a means of developing more accurate diagnostics and a means for understanding and validating the diagnostics.

The advantage to AI-ESTATE is that it provides a consistent framework for incorporating diagnostic knowledge and services in any test environment. For example, the standards provide facilities for reasoning with multiple models and for coupling fixed fault trees and dynamic EDIMs. In addition, this framework supports a "plug and play" approach for incorporating diagnostic reasoners into the ATS architecture.

5. REFERENCES

1. DePaul, R. 1985. "Logic Modeling as a Tool for Testability," *AUTOTESTCON '85 Symposium Proceedings*, Piscataway, New Jersey: IEEE Press, pp. 203-207.
2. Simpson, W. R. 1987. "Active Testability Analysis and Interactive Fault Isolation Using STAMP," *IEEE AUTOTESTCON '87 Conference Record*, Piscataway, New Jersey: IEEE Press, pp. 105-112.
3. Deb, S., Pattipati, K., and Shrestha, R. 1997, "QSI's Integrated Diagnostics Toolset," *AUTOTESTCON 97 Conference Record*, New York: IEEE Press.
4. Franco, J. 1988. "Experiences Gained Using the Navy's IDSS Weapon System Testability Analyzer," *AUTOTESTCON '88 Conference Proceedings*, Piscataway, New Jersey: IEEE Press, pp. 129-132.
5. Ben-Bassat, Beniaminy, I., M. Joseph, D. 1997. "Improving Test Strategies and Fault Isolation with Expert Systems," *AUTOTESTCON 97 Conference Record*, New York: IEEE Press.
6. IEEE Std 1232-1995. Trial Use Standard for Artificial Intelligence and Expert System Tie to Automatic Test Equipment (AI-ESTATE): Overview and Architecture.
7. IEEE Std 1232.1-1997. Trial Use Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE): Data and Knowledge Specification.
8. IEEE P1232.2. (1997, August). Draft Trial Use Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE): Service Specification, Draft 4.0.
9. ISO 10303-11:1994. Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 11: EXPRESS Language Reference Manual.
10. ISO 10303-12. Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 12: EXPRESS-I Language Reference Manual, Committee Draft.