Standardizing Diagnostic Information Using IEEE AI-ESTATE

John W. Sheppard
ARINC
2551 Riva Road
Annapolis, MD 21401
jsheppar@arinc.com

Antony Bartolini
Etec Systems, Inc.
26460 Corporate Avenue
Hayward, CA 94545
antonyb@etec.com

Leslie A. Orlidge
AlliedSignal Aerospace
Guidance and Control Systems
Teterboro, NJ 07608
leslie.orlidge@alliedsignal.com

Abstract—*The proliferation of artificially intelligent diagnostic reasoners and tools necessitates establishing standard interfaces to these tools and formal data specifications to capture relevant diagnostic information to be processed by these tools. Current test standards provide little guidance to using AI technology in test applications. Proposed AI standards (e.g., KIF) do not specifically address the concerns of the test community. Thus, no standard exists, currently, addressing the use of AI systems in test environments. AI-ESTATE is intended to fill this void. This paper will provide an update on the status of all of the AI-ESTATE standards and their potential use to support diagnostic tools and applications.*

I. Introduction

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)[1]" 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 arena of automatic test equipment, 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 tie these models to domain-specific models as needed to complete the test system.

In this paper, we describe the AI-ESTATE architecture [1] and recent progress in the standards' development. We discuss progress on defining several software services to be provided by an AI-ESTATE conformant diagnostic system [2] with emphasis given to a new information model for dynamic data. In addition, we describe initial work on a new standards project to develop a commercial replacement to MIL-STD-2165 based on the models in IEEE Std 1232.1-1997 [3]. Finally, we provide directions for future work on AI-ESTATE and issue an invitation for interested parties to participate in the AI-ESTATE development process.

II. Background

Increasing complexity and cost of current systems, the inability to consistently diagnose and isolate faults in systems using conventional means, and advances in artificial intelligence technology have fostered the growth of AI technology in test and diagnosis. The proliferation of diagnostic reasoners and tools necessitates establishing standard interfaces to these tools and formal data specifications to capture relevant diagnostic information. Current test standards (e.g., Boundary Scan and STIL) [4,5] provide no guidance for using AI technology in test applications. Proposed AI standards (e.g., KIF) do not specifically address the concerns of the test community [6]. Thus, no standard exists, currently, addressing the use of AI systems in test environments. The AI-ESTATE standards are intended to fill this void.

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.

[1] Previously, Artificial Intelligence and Expert System Tie to Automatic Test Equipment.

0-7803-4162-7/97 $5.00 © 1997 IEEE

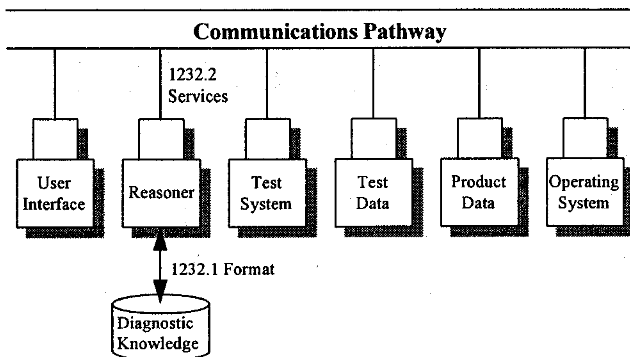

Figure 1. AI-ESTATE Architectural Concept

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.

III. An Architecture for Diagnosis

The AI-ESTATE architecture presented in Figure 1 shows a conceptual view of an AI-ESTATE-conformant system. AI-ESTATE applications may use any combination of functional elements and inter-function communication as shown in the figure. The service specification (P1232.2), 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.

As indicated in Figure 1, AI-ESTATE includes two component standards focusing on two distinct aspects of the stated objectives. The first aspect concerns the need to exchange data and knowledge between conformant diagnostic systems. By providing a standard representation of test and diagnostic data and knowledge and standard interfaces between reasoners and other elements of a test environment, test, production, operation, and support costs will be reduced.

Two approaches can be taken to address this need: providing interchangeable files (P1232.1) and providing services for retrieving the required data or knowledge through a set of standard accessor services (P1232.2). AI-ESTATE is structured such that either approach can be used [2,3].

The second aspect concerns the need for functional elements of an AI-ESTATE conformant system to interact and interoperate. The AI-ESTATE architectural concept provides for the functional elements to communicate with one another via a "communications pathway." Essentially, this pathway is an abstraction of the services provided by the functional elements to one another. Thus, implementing services of a reasoner for a test system to use results in a communication pathway being established between the reasoner and the test system.

AI-ESTATE services (P1232.2) are provided by reasoners to the other functional elements fitting within the architecture illustrated by Figure 1. These reasoners may include (but are not necessarily limited to) diagnostic systems, test sequencers, maintenance data feedback analyzers, intelligent user interfaces, and intelligent test programs. The current focus of the standards is on diagnostic reasoners. In addition to providing services to the test system, the human presentation system, a maintenance data collection system, and possibly the unit under test, the reasoner also uses services provided by these other systems as required. These services are not specified by the standards.

IV. 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, reasoner state accessor services, knowledge acquisition services, and reasoner control 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` [8]. Since knowledge acquisition services provide the create, put, and delete services to match the model accessor services, we will consider these together.

A. Static Model Traversal Services

With the publication of the data and knowledge specification [3], the first set of service defined for the service specification [2] focused on the existing models. The data and knowledge specification defines three models—a common element model, a fault tree model, and an enhanced diagnostic inference model (EDIM). The common element model provides definitions of basic entities expected to be used by any diagnostic reasoner, and the fault tree model and EDIM organize these entities in a way to facilitate diagnostic reasoning.

The model traversal services provide the means for a reasoner to process the model on line. As an example, IEEE Std 1232.1-1997 defines, using EXPRESS [7], a diagnostic model to be the following entity:

```
ENTITY diagnostic_model;
    name              :   name_type;
    description       :   description_type;
    model_test        :   SET [1:?] OF test;
    model_diagnosis   :   SET [2:?] OF diagnosis;
    model_resource    :   SET [0:?] OF resource;
    model_anomaly     :   SET [0:?] OF anomaly;
END_ENTITY;
```

Based on this model, we can define services for traversing the model to obtain information about a particular diagnostic model. For example, consider the set of tests defined in a particular model. To find the set of tests, we must first have access to the model itself. To obtain this access, we need to know either the name or the internal identifier of the model.

The service, `get_diagnostic_model(name)`, returns an identifier for a diagnostic model identified by its name. Once we have the identifier, we can then access the set of tests defined in the model with the service `get_model_tests(model_id)`. This service would return a list of tests which can then be used with another service to reference the actual test entities.

Associated with each entity in the model will be the four types of services given above (i.e., `create`, `get`, `put`, and `delete`). EXPRESS includes the ability to define functions and procedures (similar to a programming language), but the intent is for these functions and procedures to be used in defining constraints on model entities [9]. The AI-ESTATE committee observed, however, that EXPRESS provides a natural fit between information modeling and service definition since the type system is provided by the models. As a result, the committee decided to use EXPRESS to define the services as well. For example, `get_model_tests` is defined in P1232.2 as follows:

```
FUNCTION
    get_model_tests(model_id:diagnostic_model):
        SET [1:?] OF test;
END_FUNCTION;
```

Once all of the services have been defined in EXPRESS, language bindings can be provided to make the services enable the services to be implemented by a reasoner. Currently, AI-ESTATE is exploring the development of language binding for Ada and C.

B. Reasoner State Services

Given a model has been loaded and a system is being tested, the model needs to be processed for diagnosis to be performed. Simply traversing the models defined in P1232.1 is not useful since the model only defines expectations rather than reality. Any reasoner will maintain an internal state of its reasoning process that will be used to report a diagnosis or to explain its reasoning. For diagnostic reasoners to be "plug-and-play" compatible, certain diagnostic state information needs to be in common between the reasoners.

To facilitate defining services for accessing or modifying the state of the reasoner, the AI-ESTATE committee made some assumptions about common diagnostic information. These assumptions were derived from the common element model and extended to support various types of reasoners. Specifically, the AI-ESTATE committee has defined a new information model, called the *dynamic context model*, that provides the type structure for reasoner state (Figure 2).

The "starting point" for the model is given by a diagnostic `session`. The session corresponds to a sequence of steps in which the state of the reasoner changes at each `step`. At each step, the reasoner keep track of what it knows about the state of models (i.e., whether or not a model is available for processing), the resources, the tests, and the diagnoses. In addition, for each resource and test, the actual cost incurred at that step (both time and non-time) is recorded.

This model can be traversed similarly to the models defined in P1232.1. More interesting is the fact that entities can be created "on the fly" and the states of these entities vary from step to step. Another interesting byproduct of this model is that, by organizing the state according to steps in a session, the information captured can be fed to a maintenance data collection system and retained as a diagnostic history. This is the scope of the TMIMS (Standard for the Management of Test and Maintenance Information) standards project P1389. AI-ESTATE and TMIMS have been coordinating closely to ensure compatibility between their information models.

Another issue related to reasoner state is how one "explains" the reasoning process. During committee discussion, this was a highly contentious issue since no one could agree on a good method for explanation. The debate was settled when it was observed that the process of traversing reasoner state provides a rudimentary approach to explanation. Then, if tool developers want to provide "higher order" services, they can use the standard services to collect the necessary information. For example, explaining the currently inferred value for a diagnosis can be accomplished by identifying the tests in the model that affect the diagnosis and showing the step in the session where one of these tests was performed that led to the current assertion.

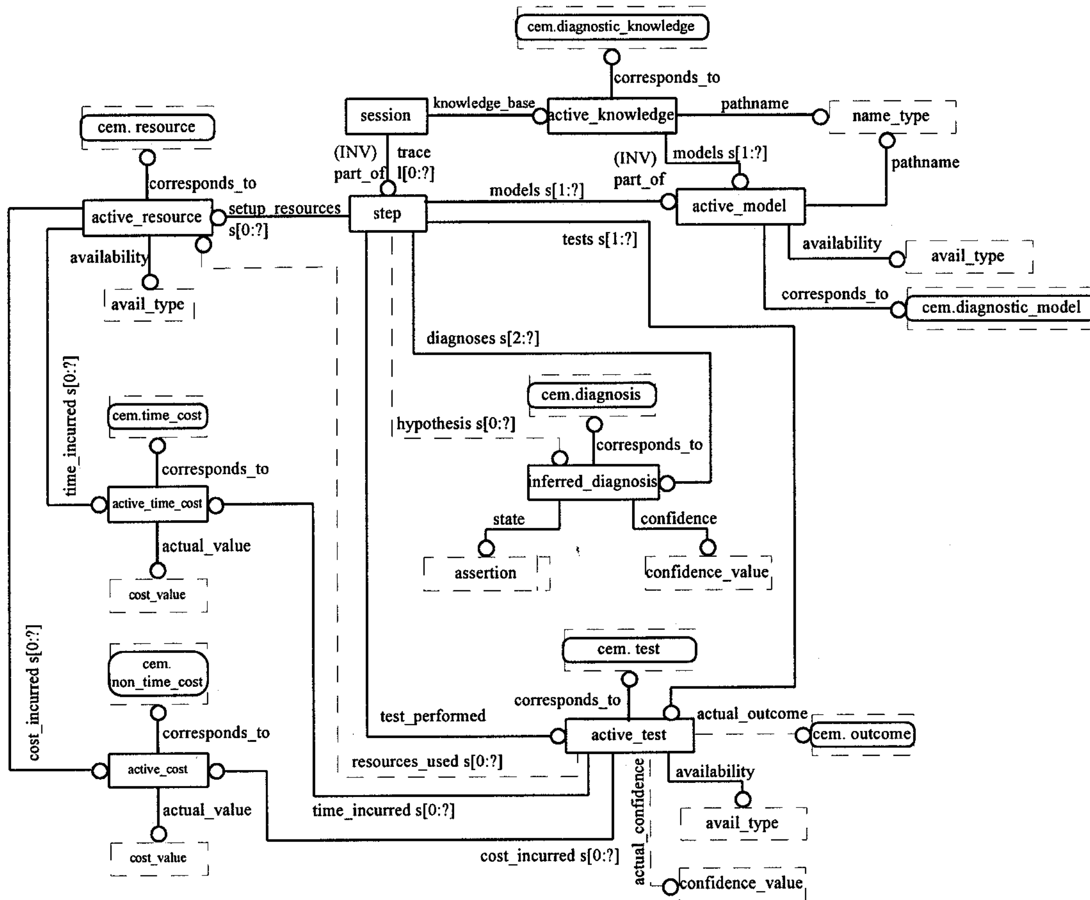

Figure 2. Dynamic Context Model

C. Reasoner Control Services

AI-ESTATE anticipates defining several services for controlling the reasoner. Currently, six services have been identified:

1. set_mode_inactive
2. set_mode_active
3. attach_model
4. detach_model
5. save_model
6. load_model

All of these services focus on making models available or updating the models.

The reasoner is assumed to be in one of three modes: null, inactive, and active. It is assumed the reasoner is in null mode until a model is attached. At that point the reasoner transitions into inactive mode automatically. Once all models have been detached, the reasoner transitions automatically back to null mode. The only way to transition between inactive and active is by a specific request from the application executive.

In a previous version of the standard, several other services were provided to control the reasoner such as setting search criteria, applying test outcomes, and reverting the state to a previous state. All of these services were deleted since they can all be handled by "putting" values in the dynamic context model. For example, applying a test outcome involves a call to `put_actual_outcome`. This would inform the reasoner that a new outcome is available for processing, and the reasoner state would be updated accordingly.

V. Testability Standards

As defined in MIL-STD-2165, testability is "a *design characteristic* which allows the status (operable, inoperable,

or degraded) of an item to be determined and the isolation of faults within the item to be performed in a timely manner [10]." The purpose of MIL-STD-2165 was to provide uniform procedures and methods to control planning, implementation, and verification of testability during the system acquisition process by the Department of Defense (DoD). It was to be applied during all phases of system development—from concept to production to fielding. This standard, though deficient in some areas, provided useful guidance to government suppliers. Further, lacking any equivalent industry standard, many commercial system developers have used it to guide their activities even though it was not imposed as a requirement.

With the current emphasis within the DoD on the use of industry standards, the continuing need to control the achievable testability of delivered systems in DoD and commercial sectors, and the removal of MIL-STD-2165 as a standard with no replacement (commercial or DoD), there is a need for a new industry standard that addresses system testability issues and that can be used by both commercial and government sectors. To be useful, this commercial standard must provide specific, unambiguous definitions of criteria for assessing system testability. In addition, the standard should recommend processes for implementing a testability program as part of a product's full life cycle.

MIL-STD-2165 was deficient in the precise definition of measurable testability figures-of-merit and relied mostly on a weighting scheme for testability assessment. (It should be noted, however, that the standard did permit the use of analytical tools for testability assessment such as SCOAP, STAMP, and WSTA.) Now that a standard diagnostic model exists [2] and the definition of reasoner services are nearing completion [3], AI-ESTATE has decided to explore creating a standard defining testability metrics in terms of the knowledge and service standards.

In the industry many terms such as test coverage and fault detection are not well defined and not comparable from system to system. Some measures, such as false alarm rate, are not measurable in field applications. An immediate benefit will come with a consistent, precise, measurable set of testability attributes that can be compared across systems and within iterations of system design.

The ability of a system implementation to support its fault isolation goals must be measured in terms of both the fundamental capability of the diagnostic model to support the reasoning process and of the ability of the reasoner to do the reasoning. For example, suppose that we have a diagnostic system that is intended to work in a time critical environment. Furthermore, suppose the diagnostic model contains the information required to fully fault isolate to the required level unambiguously. If the reasoner is too slow, then the system will not satisfy its performance requirements. Similarly, if the diagnostic model or the set of tests supporting that model contains insufficient information to achieve fault isolation to the required level, it will not matter how fast the reasoner is, the system will not satisfy its isolation requirements. Metrics must be established that allow us to clearly identify important testability parameters of both models and reasoners.

MIL-STD-2165 attempted to standardize both programmatic and measurement tasks. Variations in the internal program management methods of different companies and DoD organizations makes the standardization of the programmatic tasks unreasonable. We are exploring creating a "Recommended Practice" document to support the programmatic aspects of system testability.

MIL-STD-2165 identified five key elements of a comprehensive testability program, including the following:

1. Testability program plan
2. Establishment of achievable testability requirements
3. Participation in the design process
4. Prediction and evaluation of design testability
5. Inclusion of testability in program reviews

Lacking well defined testability measures, the tasks of establishing testability requirements, and predicting and evaluating the testability of the design are extremely difficult. This in turn makes effective participation in the design for testability process difficult. These difficulties will be greatly diminished by the establishment of standard testability metrics. Beyond the participation with design engineering MIL-STD-2165 also puts emphasis on the information exchanged between the testability and logistics functions and the testability and maintainability functions.

As we strive to establish concurrent engineering practices, the interchange between the testability function and other functions becomes even more important. To create integrated diagnostic environments, where the elements of automatic testing, manual testing, training, maintenance aids, and technical information work in concert with the testability element, we must maximize the reuse of data, information, knowledge, and software. Complete diagnostic systems include BIT, ATE, and manual troubleshooting. It would be desirable to be able to predict and evaluate the testability of systems at these levels.

Currently, the AI-ESTATE subcommittee is gathering information from the DoD and industry about model representations, their associated metrics, and the processes put in place to utilize them. The results of this review will form the basis for defining the metrics to be included in the standard and the procedural guidance to be included in the "Recommended Practice."

VI. Future Work

In spite of the large amount of work that has been done on the AI-ESTATE standards and the acceptance of the

standards in government and industry, much additional work needs to be done to keep the standards in pace with technology advances. Currently, several projects are underway within the AI-ESTATE subcommittee to do just that.

First, we recognize that there are more approaches to performing diagnosis than using fault trees and diagnostic inference models. Currently, we are in the process of defining a constraint model that will capture temporal, logical, and resource constraints in testing and diagnosing a system. Related to this is work underway in the ABBET (A Broad Based Environment for Test) subcommittee and EDIF committee defining a test requirements model (TeRM). TeRM that includes constraints for capturing information about the behavior of a product and a test resource. We expect to be able to work with the constraints defined in TeRM to facilitate reasoning with constraint-based systems.

In addition to the constraint model, we anticipate developing models for rule-based systems and for connectionist systems. Rule-based systems provided the first success stories in diagnosis and artificial intelligence. While both the EDIM and the constraint model would be able to capture the logical information contained in a rule base, a separate model is required that is tailored to the rule-based architecture.

Diagnosis occurs within some context. In fact, the context is central to determining the scope of the test and diagnosis problem. Unfortunately, capturing information about context in a standard way is problematic. The number of variables associated with context is excessive, and the relationships between those variables are frequently unknown. Nevertheless, we find that proper interpretation of diagnostic and test information relies upon a common understanding of the context in which testing takes place. As a result, we are beginning to develop a model of context for diagnosis. We believe the model capturing reasoner state provides a starting point for capturing context; however, much more work is required. Recent work in non-monotonic reasoning and categorical reasoning may offer promise in further capturing context for the diagnostic problem [11].

VIII. An Invitation

The AI-ESTATE family of standards represents the consensus of industry participants in SCC20 that standards are required for diagnostic reasoners and that these models accurately reflect the state of the practice in diagnostic tools. As such, SCC20 is constantly seeking out people in government, industry, and academia to assist in developing the standards. Anyone interested in the work of AI-ESTATE or other SCC20 standards, whether user, provider, or a generally interested party, is encouraged to get involved. Detailed information on the progress of the standards and on key personnel is available on the world wide web at "`http://www.cs.jhu.edu/~sheppard/P1232`".

VIII. Conclusion

Reasoning system technology has progressed to the point where electronic systems are employing artificial intelligence as a primary component in meeting system test and verification requirements. This is giving rise to a proliferation of AI-based design, test, and diagnostic tools. Unfortunately, the lack of standard interfaces between these reasoning systems is increasing the likelihood of significantly higher product life-cycle cost. Such costs would arise from redundant engineering efforts during design and test phases, sizable investment in special-purpose tools, and loss of system configuration control.

The AI-ESTATE standard promises to facilitate ease in production testing and long-term support of systems as well as reducing overall product life-cycle cost. This will be accomplished by facilitating portability and knowledge reuse and sharing of test and diagnostic information, among embedded, automatic, and stand-alone test systems within the broader scope of product design, manufacture, and support.

References

[1] IEEE Std 1232-1995. 1995. *Trial Use Standard for Artificial Intelligence and Expert System Tie to Automatic Test Equipment (AI-ESTATE): Overview and Architecture*, Piscataway, New Jersey: IEEE Standards Press.

[2] IEEE Std 1232.1-1997. 1997. *Trial Use Standard for Artificial Intelligence and Exchange and Service Tie to All Test Environments (AI-ESTATE): Data and Knowledge Specification*, Piscataway, New Jersey: IEEE Standards Press.

[3] IEEE P1232.2. 1997. *Trial Use Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE): Service Specification*, Draft 3.2.

[4] IEEE Std-1149.1-1990. 1990. *Standard Test Access Port and Boundary Scan Architecture*, Piscataway, New Jersey" IEEE Standards Press.

[5] IEEE P1450. 1996. *Standard Test Interface Language (STIL)*, Draft 0.23.

[6] IEEE P1252. 1993. *Standard for a Frame Based Knowledge Representation*, Draft 2.1.

[7] ISO 10303-11:1994. 1994. *Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 11: EXPRESS Language Reference Manual*, Geneva: ISO Press.

[8] Booch, G. 1994. *Object-Oriented Analysis And Design With Applications*, 2nd Ed. Benjamin Cummings.

[9] Schenk, D. A. and P. R. Wilson. 1994. *Information Modeling: The EXPRESS Way*, New York: Oxford University Press.

[10] MIL-STD-2165. 1985. *Testability Requirements for Electronic Systems and Equipments.*

[11] Akman, V. and M. Surav. 1996. "Steps Toward Formalizing Context," *AI Magazine*, 17(3):55–72.