

IEEE Std 1232™-2002
(Revision of IEEE Std 1232-1995,
IEEE Std 1232.1™-1997, and
IEEE Std 1232.2™-1998)

1232™

IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE)

IEEE Standards Coordinating Committee 20

Sponsored by the
IEEE Standards Coordinating Committee 20



Published by
The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

22 November 2002

Print: SH95000
PDF: SS95000

Recognized as an
American National Standard (ANSI)

IEEE Std 1232™-2002
(Revision of IEEE Std 1232-1995,
IEEE Std 1232.1-1997, and
IEEE Std 1232.2-1998)

IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATES)

Sponsor

IEEE Standards Coordinating Committee 20

Approved 13 November 2002

American National Standards Institute

Approved 13 June 2002

IEEE-SA Standards Board

Abstract: AI-ESTATE is a set of specifications for data interchange and for standard services for the test and diagnostic environment. The purpose of AI-ESTATE is to standardize interfaces between functional elements of an intelligent diagnostic reasoner and representations of diagnostic knowledge and data for use by such diagnostic reasoners. Formal information models are defined to form the basis for a format to facilitate exchange of persistent diagnostic information between two reasoners, and also to provide a formal typing system for diagnostic services. This standard then defines the services to manipulate diagnostic information and to control a diagnostic reasoner.

Keywords: AI-ESTATE, diagnosis, diagnostic interference, diagnostic model, diagnostic services, dynamic content, fault tree, knowledge exchange, system test

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2002 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 22 November 2002. Printed in the United States of America.

Print: ISBN 0-7381-3295-0 SH95000
PDF: ISBN 0-7381-3296-9 SS95000

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Associations (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability of fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "AS IS."

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owned by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923, USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

[This introduction is not part of IEEE Std 1232-2002, IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE).]

This AI-ESTATE standard provides a formal framework for exchanging diagnostic knowledge and constructing diagnostic reasoners. The intent is to provide a standard framework for identifying required information for diagnosis and defining the diagnostic information in a machine-processable way. In addition, software interfaces are defined whereby diagnostic tools can be developed to process the diagnostic information in a consistent and reliable way.

Participants

At the time that this standard was completed, the Diagnostic and Maintenance Control (D & MC) Subcommittee had the following membership:

John W. Sheppard, *Co-Chair*

Mark A. Kaufman, *Co-Chair*

Timothy M. Bearse
Gregory P. Bowman
Antonius Bunsen
Bernard Dathy
Somnath Deb
Bernd Dinklage
David B. Droste
Sudipto Ghoshal

Amanda Jane Giarla
Eric S. Gould
Arnold M. Greenspan
David L. Kleinman
Franc Novak
Leslie A. Orlidge
Jean Pouilly

William C. Rodriguez
William R. Simpson
Philip T. Smith
Li Pi Su
Jack Taylor
Joerg H. Urban
J. Richard Weger
Timothy J. Wilmering

In addition, these individuals have contributed review and comments:

Antony Bartolini
Larry Howard

Michael Lynch

William Simerly
Lee A. Shombert

The following members of the balloting group voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Guy Adam
Gregory P. Bowman
Joseph F. Buch
Eric M. Bukata
Danny C. Davis
Somnath Deb
Guru Dutt Dhingra
William Brit Frank
Sudipto Ghoshal

Amanda Jane Giarla
Eric S. Gould
Osamu Karatsu
Mark A. Kaufman
Frank X. Kearns
Leslie A. Orlidge
Jean Pouilly
Narayanan Ramachandran
David E. Rolince
Eric Sacher

John W. Sheppard
Lee A. Shombert
William R. Simpson
Philip T. Smith
Joseph J. Stanco
William John Taylor
Kirk D. Thompson
Timothy J. Wilmering
Jehuda Ziegler

When the IEEE-SA Standards Board approved this standard on 13 June 2002, it had the following membership:

James T. Carlo, *Chair*

James H. Gurney, *Vice Chair*

Judith Gorman, *Secretary*

Sid Bennett
H. Stephen Berger
Clyde R. Camp
Richard DeBlasio
Harold E. Epstein
Julian Forster*
Howard M. Frazier
Toshio Fukuda

Arnold M. Greenspan
James H. Gurney
Raymond Hapeman
Donald M. Heirman
Richard H. Hulett
Lowell G. Johnson
Joseph L. Koepfinger*

Peter H. Lips
Nader Mehravari
Daleep C. Mohla
William J. Moylan
Malcolm V. Thaden
Geoffrey O. Thompson
Howard L. Wolfman
Don Wright

*Member Emeritus

Also included is the following nonvoting IEEE-SA Standards Board liaison:

Alan H. Cookson, *NIST Representative*
Satish K. Aggarwal, *NRC Representative*

Savoula Amanatidis,
IEEE Standards Managing Editor

Contents

1. Overview.....	1
1.1 Scope.....	2
1.2 Purpose	2
1.3 Conventions used in this document.....	3
2. References	3
3. Definitions and acronyms	3
3.1 Definitions.....	3
3.2 Acronyms	5
4. Description of AI-ESTATE	5
4.1 AI-ESTATE architecture	5
4.2 Interchange format.....	8
4.3 Binding strategy	8
4.4 Extensibility.....	9
4.5 Status codes.....	10
4.6 Conformance.....	11
4.7 Service order dependence.....	12
5. Models.....	15
5.1 Common Element Model.....	15
5.2 Diagnostic Inference Model.....	48
5.3 Dynamic Context Model	52
5.4 Enhanced Diagnostic Inference Model.....	83
5.5 Fault Tree Model.....	87
6. Services.....	91
6.1 Model management services	92
6.2 Reasoner manipulation services.....	97
Annex A (informative) Bibliography	105
Annex B (informative) Overview of EXPRESS.....	107

IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE)

1. Overview

The Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE) standard was developed by the Diagnostic and Maintenance Control (D & MC) Subcommittee of the IEEE Standards Coordinating Committee 20 (SCC 20) on Test and Diagnosis for Electronic Systems to serve as a standard for the application of artificial intelligence to system test and diagnosis. This AI-ESTATE standard defines interfaces among reasoners and reasoning system users, test information knowledge bases, and more conventional databases. In addition to interface standards, the AI-ESTATE standard includes a set of formal data specifications to facilitate the exchange of system under test related diagnostic information.

This standard describes a set of formal data and knowledge specifications consisting of the logical representation of devices, their constituents, the failure modes of those constituents, and tests of those constituents. The data and knowledge specification provides a standard representation of the common data elements required for system test and diagnosis. This will facilitate portability of test-related knowledge bases for intelligent system test and diagnosis.

The goals of this standard are summarized as follows:

- Incorporate domain specific terminology
- Facilitate portability of diagnostic knowledge
- Permit extensibility of diagnostic knowledge
- Enable the consistent exchange and integration of diagnostic capabilities

This standard provides a controlled extension mechanism to allow inclusion of new diagnostic technology outside the scope of the AI-ESTATE specification.

One of the purposes of this standard is to define information models for knowledge bases to be used in the context of test and diagnosis and, from these models, to derive a data interchange format. The specifications in this standard shall support fully portable diagnostic knowledge. No host computer dependence is contained in the AI-ESTATE standard.

AI-ESTATE defines key data and knowledge specification formats. Implementations that use only these specification formats will be portable. This does not preclude use of AI-ESTATE interfaces with nonconformant specification formats; however, such implementations may not be portable. As shown in Figure 1, a diagnostic model can be moved from one AI-ESTATE implementation to another by translating it into the interchange format. Another AI-ESTATE implementation can then utilize this information as a complete package by translating the data and knowledge from the interchange format to its own internal form.

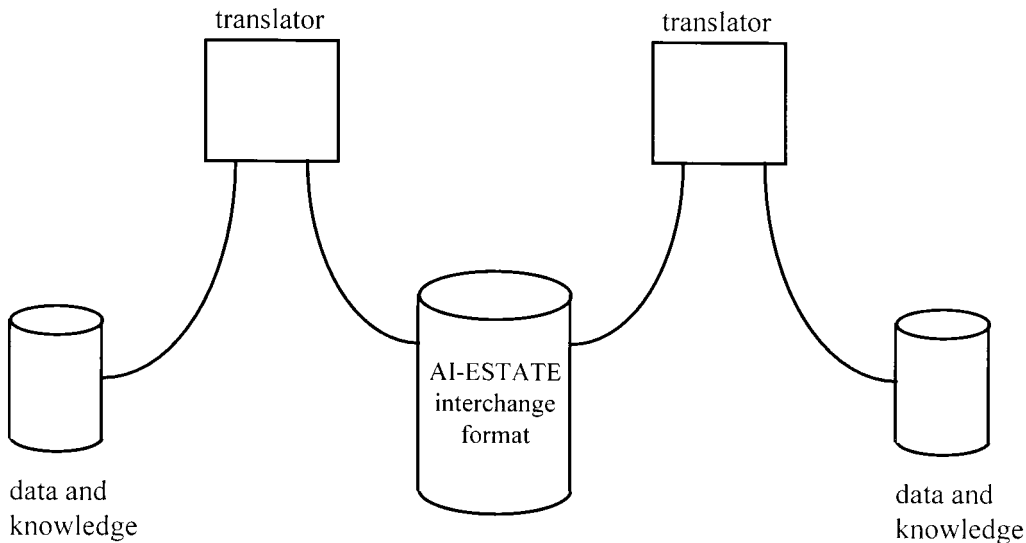


Figure 1—An example of AI-ESTATE’s portability mechanism for data and knowledge

The translation step is not a requirement; an AI-ESTATE implementation may use the interchange format for its own internal form.

Software specifications defined in this standard will ensure the interchangeability of diagnostic reasoners through the definition of encapsulated services. This will allow diagnostic reasoners to be interchanged within an AI-ESTATE conformant system with no effect on the other elements of the system.

1.1 Scope

The AI-ESTATE standard defines formal specifications for supporting system diagnosis. These specifications support the exchange and processing of diagnostic information and the control of diagnostic processes. Diagnostic processes include, but are not limited to, testability analysis, diagnosability assessment, diagnostic reasoning, maintenance support, and diagnostic maturation.

1.2 Purpose

The AI-ESTATE standard provides formal models of diagnostic information to ensure unambiguous access to an understanding of the information supporting system test and diagnosis. The standard unifies and expands on the specifications published in IEEE Std 1232TM-1995 [B3],¹ IEEE Std 1232.1TM-1997 [B4], and IEEE Std 1232.2TM-1998 [B5].

¹The numbers in brackets correspond to those of the bibliography in Annex A.

1.3 Conventions used in this document

This standard specifies information models using the EXPRESS language and uses the following conventions in their presentation:

All specifications in the EXPRESS language are given in the `Courier` type font. This includes references to entity and attribute names in the supporting text. The EXPRESS models found in Clause 5 include comment delimiters “(*)” and “(“),” thus allowing extraction of the models from an electronic version of the standard for direct use.

Each entity of each EXPRESS schema is presented in a separate subclause. Within a schema, subclauses are listed in alphabetical order by constants, types, enumerated types, select types, entities, and then functions. The subclause structure begins with the actual EXPRESS specification, then each attribute of the entity is described below the attribute definition heading. If any constraints have been specified, these are described below the formal propositions heading.

This standard uses the vocabulary and definitions of relevant IEEE standards. In case of conflict of definitions, the following precedence shall be observed: 1) AI-ESTATE definitions (Clause 3); 2) SCC20 documentation and standards; and 3) IEEE 100TM, *The Authoritative Dictionary of IEEE Standards Terms*, Seventh Edition [B2].

Clause 6 of this standard presents the formal specification of the encapsulated services of this standard. EXPRESS is used to represent the interface of each individual service defining the semantics and type of the required value to be returned.

2. References

This standard shall be used in conjunction with the following publications.

ISO 10303-11:1994, Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 11: Description Methods: The EXPRESS Language Reference Manual.²

ISO 10303-21:1994, Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 21: Implementation Method: Clear Text Encoding of the Exchange Structure.

3. Definitions and acronyms

For the purposes of this standard, the following terms and definitions apply. IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms*, Seventh Edition [B2], should be referenced for terms not defined in this clause.

3.1 Definitions

3.1.1 ambiguity: In fault isolation, the inability to localize to a single diagnosis for a repair level, given a set of test results, observations, or other information.

3.1.2 ambiguity group: The collection of all diagnoses that are in ambiguity.

²ISO publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse (<http://www.iso.ch/>). ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA (<http://www.ansi.org/>).

3.1.3 application executive: A software component (or the role of a software component) that administers, and coordinates services to other components within a test system.

3.1.4 diagnosis: The conclusion(s) inferred from tests, observations, or other information.

3.1.5 diagnostic procedure: A structured sequence of tests, observations, and other information used to localize a fault or faults.

3.1.6 element: The smallest entity of an Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE) model. For example, in a particular model, the smallest test, the smallest diagnosis, and the no-fault conclusion are all elements.

3.1.7 failure: The loss of ability of a repair item, equipment, or system to perform a required function. The manifestation of a fault. Within AI-ESTATE models, a manifestation is given by the outcome of a test.

3.1.8 false alarm: An indicated fault where no fault exists.

3.1.9 fault: A defect or flaw in a hardware or software component.

3.1.10 fault isolation: The process of reducing the set of diagnoses in ambiguity to a degree sufficient to undertake an appropriate corrective action.

3.1.11 fault localization: The reduction of ambiguity by the application of tests, observations, or other information.

3.1.12 interoperability: The ability of two or more systems or elements to exchange information and to use the information that has been exchanged.

3.1.13 knowledge base: A combination of structure, data, and function used by reasoning systems.

3.1.14 level of maintenance: A level at which test, diagnosis, and repair operates (e.g., maintenance depot, factory, in the field).

3.1.15 portability: The capability of being moved between systems.

3.1.16 protocol: A set of conventions or rules that govern the interactions of processes or applications within a computer system or network.

3.1.17 reasoning system: A system that can combine elements of information and knowledge to draw conclusions.

3.1.18 replaceable unit: A collection of one or more parts considered as a single part for the purposes of replacement and repair due.

3.1.19 resource: Any capability that is to be scheduled, assigned, or controlled by the underlying implementation to assure nonconflicting usage by processes.

3.1.20 service: A software interface providing a means for communicating information between two applications. An action or response initiated by a process (i.e., server) at the request of some other process (i.e., client).

3.1.21 system: 1) A collection of interacting, interrelated, or interdependent elements forming a collective, functioning entity; 2) a set of objects or phenomena grouped together for classification or analysis; 3) a collection of hardware or software components necessary for performing a high-level function.

3.1.22 test: A set of stimuli, either applied or known, combined with a set of observed responses and criteria for comparing these responses to a known standard.

3.1.23 test strategy: An approach taken to combine factors including constraints, goals, and other considerations to be applied to the testing of a system under test.

3.2 Acronyms

AI-ESTATE	Artificial Intelligence Exchange and Service Tie to All Test Environments
BIT	Built-In Test
CEM	Common Element Model
DCM	Dynamic Context Model
DIM	Diagnostic Inference Model
EDIM	Enhanced Diagnostic Inference Model
FTM	Fault Tree Model
SNMP	Simple Network Management Protocol
UTC	Coordinated Universal Time

4. Description of AI-ESTATE

4.1 AI-ESTATE architecture

This standard provides the following:

- Overview of the AI-ESTATE architecture
- Formal definition of diagnostic models for systems under test
- Formal definition of encapsulated software services for diagnostic reasoners

AI-ESTATE focuses on two distinct aspects of the stated purpose. The first aspect concerns the need to exchange data and knowledge between conformant systems. Two approaches can be taken to address this need: providing interchangeable files and providing services for retrieving the required data or knowledge through an information management system. AI-ESTATE is structured such that either approach can be used. 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 “communication pathway” as depicted in Figure 2. Essentially, this pathway is an abstraction of the services provided by the functional elements to one another. Thus, the implemented services provide a communication pathway between the reasoner and the rest of the test system.

Services are provided by reasoners to the other functional elements of an AI-ESTATE conformant system. Reasoners can include (but are not necessarily limited to) diagnostic systems, test sequencers, maintenance data feedback analyzers, intelligent user interfaces, and intelligent test programs. AI-ESTATE will not specify services between functional elements that do not incorporate artificial intelligence capabilities. Thus, services are provided by a reasoner to the test system, the human presentation system, a maintenance data/knowledge collection system, and possibly the system under test. The reasoner shall use services provided by these other systems as required. Note that these services shall not be specified by the AI-ESTATE standard.

Data interchange formats are specified to provide a means for exchanging knowledge bases between conformant systems without the need to apply an information management system. Recognizing that

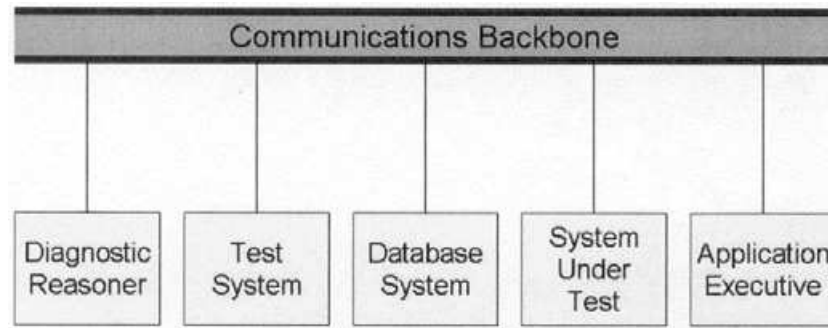


Figure 2—AI-ESTATE architectural concept

some applications may provide services for extracting required data or knowledge, services are specified to permit an application to query the diagnostic system for this purpose.

This standard facilitates the use of standard representations of diagnostic data and knowledge within the context of an AI-ESTATE implementation. In specifying data and knowledge for these domains, a structure has been constructed, as shown in Figure 3. At the top level is the Common Element Model that specifies elements common to the AI-ESTATE domain of equipment test and diagnosis in its entirety. Examples of common element constructs are *diagnosis* (diagnostic conclusions about the system under test), *repair_item* (the physical entity being repaired), *resource*, and *test*. These constructs are characterized by attributes such as costs and failure rates, which are also specified in the Common Element Model.

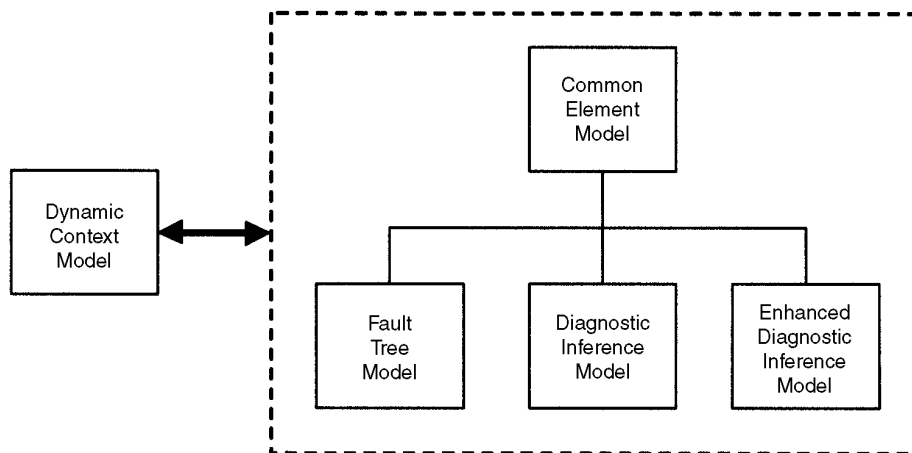


Figure 3—Hierarchical structure of AI-ESTATE models

Below this top layer is a layer of application-specific data and knowledge formats (i.e., the Diagnostic Inference Model, the Enhanced Diagnostic Inference Model, and the Fault Tree Model). These models take advantage of the constructs in the Common Element Model and tailor the constructs to the application's particular reasoning requirements.

Other data and knowledge specification formats are envisioned and will be included in future revisions of this standard. Examples include a constraint-based model, a Bayesian network model, and a neural network model. The Common Element Model has been specified such that other data and knowledge specification formats can also utilize its constructs as base elements that are tailored to the particular application's needs.

This standard also defines the encapsulated services to be provided by a diagnostic reasoner in an AI-ESTATE conformant implementation. All of the basic services are defined relative to the entities and attributes of the information models. These services can be thought of as reasoner responses to client requests from the other components of the system architecture, which include the human presentation (or user interface) system, the maintenance data/knowledge collection (or information management) system, other reasoners, and test controller(s), as shown in Figure 4. As can be seen in Figure 4, each of the elements that interface with the reasoner will provide their own set of encapsulated services to their respective clients, but those service definitions are beyond the scope of this document.

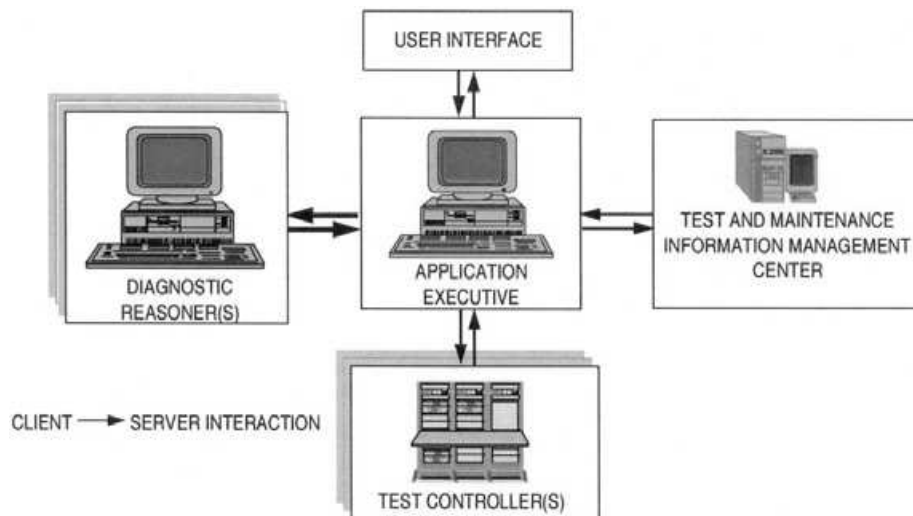


Figure 4—AI-ESTATE client server view

The definition of these services was performed by first creating accessor services for the static information models. As the specification of these services became clearer, it became apparent that to define useful services adequately for an active diagnostic reasoner new entities would have to be defined. These entities represent the context within which reasoning is performed and thereby maintain the state of the diagnostic process. Thus evolved the Dynamic Context Model as specified in 5.3 of this standard. Model management and reasoner manipulation services are defined in Clause 6 to provide a more manageable interface for interacting with the reasoner.

The definition of encapsulated services provides a means for hiding the details of any particular reasoner implementation. Such services encompass an *abstraction* of that behavior which is common to all diagnostic reasoners, regardless of implementation details. Therefore, it is the mechanism of encapsulation that provides for the interchangeability of diagnostic reasoners within an AI-ESTATE conformant system.

It should be noted that to be conformant, a reasoner implementation should provide, at the least, a status indicator (see 4.5) as a response to any service request defined by this specification. The diagnostic reasoner shown in Figure 5 becomes AI-ESTATE conformant when it provides the services specified by this standard to the application executive client.

In defining the behavior of the diagnostic reasoner, it is reasonable to expect applications to be developed whereby multiple models are used by a single reasoner or multiple reasoners interact. For the latter, it is assumed that the application executive will be responsible for managing these reasoners and reconciling information provided by the reasoners. For the single reasoner, it is assumed that the reasoner will manage the models and that this management will be based on definition of a consistent

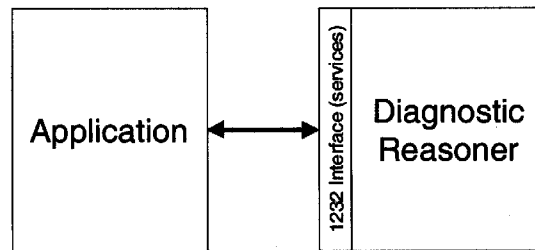


Figure 5—IEEE Std 1232-2002 interface layer

name-space with proper scoping defined by the models' contexts. For reasoners, responsible for merging information in multiple models (or for editing tools that merge models), it is assumed that such a merger will be conducted either through low-level model management services or through some proprietary mechanism. This standard does not define how model merging is to be accomplished, and no specific merger service is provided.

4.2 Interchange format

AI-ESTATE models are intended to facilitate data interchange in the context of test and diagnosis. The data interchange format will permit exchange of diagnostic models, using a neutral standard format, thus providing portability of diagnostic knowledge across applications. The interchange format used for the models defined in this standard is derived from ISO 10303-21:1994. Given this exchange structure, it is assumed that one and only one model will be contained in any given file.

The version of this standard used shall be noted in the header of the exchange file, e.g., IEEE Std 1232-2002.

4.3 Binding strategy

The intent of the binding strategy is to guide software developers in the creation of a binding layer that will expose an interface that matches the interface of the AI-ESTATE services as they are specified in this standard. The binding layer will thus insulate the application and the diagnostic reasoner from any non-AI-ESTATE details such as connectivity technology, memory management, etc.

An AI-ESTATE software system will consist of at least two components—the application and a diagnostic reasoner. The diagnostic reasoner will present an interface conformant to IEEE Std 1232-2002; the application will use AI-ESTATE services as needed by calls to this interface.

For each AI-ESTATE service there will be a corresponding function in the binding layer that will be written in the implementation language. The interfaces provided by the functions should correspond exactly to the interfaces of the services they implement (or as closely as possible given the constraints of the implementation language). All other details should be hidden from the client. This implies that the binding layer provides data-type definitions that correspond to the types specified in this standard.

The application and diagnostic reasoner programs can be written in different languages as long as the translation is handled transparently to the two programs, i.e., in the binding layer or lower. When publishing the interface, it is recommended that documentation of traceability of the elements of the interface to the services specified in the standard be provided.

For example, consider the `get_name` service as specified in EXPRESS:

```
FUNCTION get_name
  (ID_entity : entity_id):
    name_type;
END_FUNCTION;
```

It has the name `get_name`, accepts one argument of `entity_id`, and returns a `name_type`. The declaration of a corresponding binding function written in C would be

```
name_type get_name (entity_id ID_entity);
```

This might exist in a C header file and would provide the client code with an interface corresponding exactly to that of the EXPRESS form. For example,

```
name_type get_name (entity_id ID_entity)
{
  name_type name;
  :
  :
  name = ...; /* Could be a call to a function in the */
              /* server-side binding layer. */
  :
  :
  return name;
}
```

The following C data types could be defined to correspond to the AI-ESTATE types:

```
typedef char*    name_type;
typedef void*    entity_id;
```

For pure object-oriented languages such as Java™, the interface shall be presented as methods in objects. It is suggested that the information model be used to start building the class hierarchy.

4.4 Extensibility

4.4.1 Extending information models

The data and knowledge specification should be sufficiently flexible to allow for differences in the common tools that might use it, yet also remain formal and rigid enough to support porting of an AI-ESTATE exchange file to any AI-ESTATE conformant application. This flexibility is achieved by using the EXTEND schema. In general, an extended schema will not be usable by other AI-ESTATE conformant applications. In applying the EXTEND schema, the following rules shall apply:

- a) An IEEE Std 1232-2002 information model can be extended for any single model, hereafter referred to as the *extended model*, by inclusion of one or more schemata that define all entities used in the extended model. The schemata shall be legal EXPRESS and shall be syntactically and semantically correct. Further, extensions shall not be used to rename or redefine data in the standard schemata.
- b) Data contained in a model definition with an EXTEND schema shall include only entity types that are declared in IEEE Std 1232-2002, or are legal, previously defined extension entities, as defined in subsequent requirements.
- c) An EXTEND schema entity shall be declared to be a subtype of an entity defined in IEEE Std 1232-2002 or a previously defined EXTEND schema entity.

- d) A model defined with EXTEND schemata shall not cause an application to reach an invalid conclusion when processing the model with the extended entities removed. In other words, any conclusion drawn by a conformant reasoner using an extended model without the extended entities will be consistent with the information contained in the nonextended model. Further, consistency cannot be achieved through extension.
- e) All schema, entity, type, function, and procedure identifiers declared within the EXTEND schema shall begin with the prefix “extend_.”
- f) The EXTEND schema shall use the EXPRESS reference interface specification, “REFERENCE FROM,” to allow visibility of the entities within the EXTEND schema.

The following is an example of EXTEND schema:

```

SCHEMA extend_schema_a;

REFERENCE FROM common_element_model (outcome, cost);

TYPE extend_attribute_type1 = STRING;
END_TYPE;

TYPE extend_attribute_type2 = REAL;
END_TYPE;

ENTITY extend_ent_a
    SUBTYPE OF (cost);
    att1 :    extend_attribute_type1;
    att2 :    extend_attribute_type2;
END_ENTITY;

ENTITY extend_ent_b
    SUPERTYPE OF (extend_ent_c);
    SUBTYPE OF (outcome);
    att3 :    extend_attribute_type1;
END_ENTITY;

ENTITY extend_ent_c
    SUBTYPE OF (extend_ent_b);
    att4 :    extend_attribute_type1;
    att5 :    SET OF [0:?] cost;
END_ENTITY;

END_SCHEMA;

```

4.4.2 Extending services

Any application can provide services beyond those defined in this standard; however, such services will not be recognized as conforming to the standard.

4.5 Status codes

An AI-ESTATE diagnostic reasoner shall provide a means for its clients to determine the success or failure of service requests. This shall be implemented by means of the exception

entity within the Dynamic Context Model. This entity will assume one of the following mnemonic values:

```

Operation_completed_successfully
Nonexistent_data_element_requested
Missing_or_invalid_argument
Operation_out_of_sequence
Invalid_model_schema
Service_not_available
Unknown_exception_raised

```

The actual mapping will be provided by the implementation.

4.6 Conformance

This subclause defines the requirements for conformance with IEEE Std 1232-2002. It defines the minimum capabilities that are required and what subsets and extensions are allowed. This standard specifies minimal conformance for IEEE Std 1232-2002 conformant model development systems and IEEE Std 1232-2002 conformant application runtime systems. It is expected that actual implementations will conform at levels beyond the minimal levels.

Extensions to the information models are only allowed through the facility of the EXTEND schema. An EXTEND schema shall conform to the formal EXPRESS syntax and semantics found in ISO 10303-11:1994. These extensions shall be clearly identified as specified in 4.4 and should be submitted to the IEEE D & MC subcommittee. An implementation may extend the data and knowledge specification defined in this and subsequent clauses provided that the implementation clearly identifies what extensions have been implemented.

For an application to be IEEE Std 1232-2002 conformant, the specific elements of the standard to which the application conforms shall be specified and published using the matrix in Table 1. Minimal requirements for filling out this matrix are provided in 4.6.1 and 4.6.2.

Core model elements are those elements in the information model that exist as required attributes of some other required entity within the information model. Enhanced model elements are those elements in the information model that exist as optional attributes of required entities or as any kind of attribute of optional entities within the information model. Core model management services are those services that manipulate core entities in the information models. Enhanced model management services are those services that manipulate enhanced entities in the information model. Core reasoner

Table 1—Conformance matrix format

		CEM	FTM	DIM	EDIM	DCM
Model exchange	Read core model elements					
	Write core model elements					
	Read enhanced model elements					
	Write enhanced model elements					
Application services	Core model management services					
	Enhanced model management services					
	Core reasoner manipulation services					
	Enhanced reasoner manipulation services					

services are those reasoner manipulation services listed within the standard as “required,” and enhanced reasoner services are those reasoner manipulation services listed within this standard as “optional.”

To claim conformance to core model element exchange, at a minimum the core models shall include at least one of the FTM, DIM, EDIM, or DCM *in addition to* the CEM.

To claim conformance to core model management services, at a minimum the core services shall include at least those for one of the FTM, DIM, EDIM, or DCM *in addition to* those for the CEM.

To claim conformance to enhanced model element exchange, the corresponding core model exchange conformance shall also be claimed. When claiming conformance to enhanced model exchange, the specific enhanced elements shall be listed.

To claim conformance to reasoner manipulation services, one indicates conformance under the DCM.

To claim conformance to the enhanced services, the corresponding core service conformance shall also be claimed. When claiming conformance to enhanced services, the specific services shall be listed.

Note that certain services that operate on elements of the DCM depend on the reasoner being in the proper state and may depend on other services having been performed first. State dependence tables are provided in 4.7 (Table 4 and Table 5).

4.6.1 Minimal model development conformance

To claim minimal conformance to IEEE Std 1232-2002 for model development, a conformance matrix containing at least the following shall be provided, as shown in Table 2.

Table 2—Sample conformance model matrix

	CEM	FTM	DIM	EDIM
Read/write core model elements	All model elements	All model elements for at least one model shall be specified		
Read/write enhanced model elements	Specify which model elements	N/A ^a		

^aNot applicable.

4.6.2 Minimal runtime environment conformance

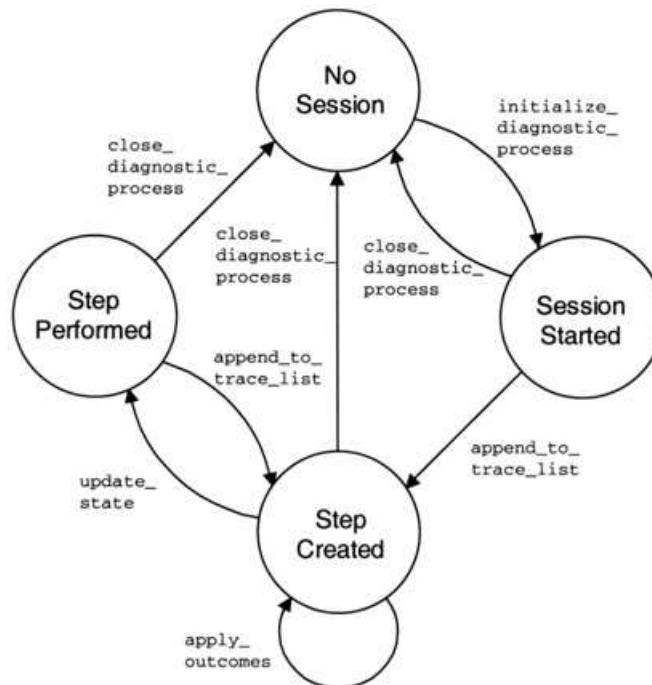
To claim minimal conformance to IEEE Std 1232-2002 for the application runtime environment, a conformance matrix containing at least the items listed in Table 3 shall be provided.

4.7 Service order dependence

The basic execution model for an AI-ESTATE conformant diagnostic reasoner assumes the ability to manipulate the Dynamic Context Model (DCM) to maintain reasoner state and follows the high-level state diagram shown in Figure 6.

Table 3—Sample runtime conformance matrix format

	CEM	FTM	DIM	EDIM	DCM
Core model management services	All services	All services for at least one model shall be specified			All services
Enhanced model management services	Specify which services	N/A ^a	Specify which services	Specify which services	Specify which services
Core reasoner manipulation services	N/A ^a				All services
Enhanced reasoner manipulation services	N/A ^a				Specify which services

^aNot applicable.**Figure 6—Execution model for an AI-ESTATE conformant diagnostic reasoner**

Initially, the reasoner will be in the state indicated by “No Session.” From this state, a process is started wherein a session within the DCM is created. Upon initiation of the process, the state transitions to “Session Started.” At this point, for reasoning to occur, the reasoner creates a step within the DCM and attaches that step to the session. The state then transitions to “Step Created.” From this state, tests are selected, sent to the test system or monitored from a test system, and test information is captured. Once the test information is captured, the resulting test outcomes are applied and the state of the DCM is updated to reflect any inferences drawn. At the same time, the state transitions to “Step Performed.” If additional testing is required, another step is created, and the state transitions back to “Step Created.” Otherwise, the session and diagnostic process can be closed, and the state returns to “No Session.” Note that at any point in the process, the session can be closed with the state transitioning directly to “No Session.”

Throughout a diagnostic session, the current step in the session shall be identified as the last instantiated step in that session. All services related to the DCM are executed within the context of the associated session. For a given reasoner, one and only one DCM can be instantiated at a time.

The availability of services is dependent on the reasoner state at any given time. For instance, while reasoning one cannot alter models other than the DCM. The definition of legal states and order dependence of the services defined within this standard are given in the service dependence matrix, shown in Table 4 and Table 5.

Table 4—Core model (CEM, FTM, DIM, EDIM) management services

Service	State	Subclause	Note
create_(entitytype)	no-session	6.1.1	
delete (ID_(entitytype) : entity_id)	no-session	6.1.2	
get_(attributename)	all	6.1.3	
put_(attributename)	no-session	6.1.4	
get_(attribute_name)_count (ID_entity : entity_id)	no-session	6.1.5	
does_(attribute_name)_exist (ID_entity : entity_id)	no-session	6.1.6	
get_nth_(attribute_name)	all	6.1.7.1	
get_last_(attribute_name)	all	6.1.7.2	
put_nth_(attribute_name)	no-session	6.1.7.3	
put_last_(attribute_name)	no-session	6.1.7.4	
insert_in_(attributename)_list	~no-session ^a	6.1.8.1	
append_to_(attributename)_list	~no-session ^a	6.1.8.2	
remove_from_(attributename)_list	~no-session ^a	6.1.8.3	
Add_to_(attributename)_set	no-session	6.1.9.1	
remove_from_(attributename)_set	no-session	6.1.9.2	
get_type (attributename)	all	6.1.10	

^aThe designation “~no-session” means all states other than “no-session”.

Table 5—Dynamic core model management services. A state dependence column is omitted since all services are available in all states

Service	Subclause
create_(entitytype)	6.1.1
delete (ID_(entitytype) : entity_id)	6.1.2
get_(attributename)	6.1.3
put_(attributename)	6.1.4
get_(attribute_name)_count (ID_entity : entity_id)	6.1.5
does_(attribute_name)_exist (ID_entity : entity_id)	6.1.6
get_nth_(attribute_name)	6.1.7.1
get_last_(attribute_name)	6.1.7.2
put_nth_(attribute_name)	6.1.7.3
put_last_(attribute_name)	6.1.7.4
insert_in_(attributename)_list	6.1.8.1
append_to_(attributename)_list	6.1.8.2
remove_from_(attributename)_list	6.1.8.3
add_to_(attributename)_set	6.1.9.1
remove_from_(attributename)_set	6.1.9.2
get_type (attributename)	6.1.10

5. Models

This clause contains the specifications for all of the information models used within an AI-ESTATE framework. Each of the models is defined using EXPRESS. A brief overview of EXPRESS and EXPRESS-G can be found in Annex B of this standard.

```
(* EXPRESS Specification starts here. *)

( *
```

5.1 Common Element Model

The Common Element Model permits the definition of the form and relationships of systems under test and tests at their most basic level. The diagnosis entity corresponds to a diagnostic conclusion about the system under test. Tests are represented as test entities within the model.

EXPRESS specification:

```
*)
SCHEMA AI_ESTATE_COMMON_ELEMENT_MODEL;
( *
```

5.1.1 Constant

EXPRESS specification:

```
*)
  CONSTANT
    no_fault      :name_type:= 'No Fault';
  END_CONSTANT;
( *
```

5.1.2 bound

This defines a type for specifying upper and lower bounds for numeric-based entities.

EXPRESS specification:

```
*)
  TYPE bound = REAL;
  END_TYPE;
( *
```

5.1.3 confidence_value

This defines a type for specifying “default” confidence values. For most uncertainty-based systems, default confidence is typically taken to be some number near 1.0.

EXPRESS specification:

```
* )
  TYPE confidence_value = REAL;
  WHERE
    range      : (0.0 <= SELF) AND (SELF <= 1.0);
  END_TYPE;
( *
```

Formal propositions:

range : The range of legal values for confidence is restricted, and the actual value is restricted to lie within this legal range.

5.1.4 cost_value

This defines a type for associating predicted values with a numeric attribute.

EXPRESS specification:

```
* )
  TYPE cost_value = REAL;
  END_TYPE;
( *
```

5.1.5 description_type

This type defines means for associating descriptive text with an entity within the model.

EXPRESS specification:

```
* )
  TYPE description_type = STRING;
  END_TYPE;
( *
```

5.1.6 name_type

This type defines a means for associating an identifying name with an entity in the model.

EXPRESS specification:

```
* )
  TYPE name_type = STRING;
  END_TYPE;
( *
```

5.1.7 non_negative_integer

EXPRESS specification:

```

*)
  TYPE non_negative_integer = INTEGER;
  WHERE
    non_negative: (SELF >= 0);
  END_TYPE;
( *

```

Formal propositions:

non_negative : Ensures that the value is not less than zero.

5.1.8 cost_type

This defines a type for categorizing a cost attribute used to optimize the diagnostic process.

EXPRESS specification:

```

*)
  TYPE cost_type = ENUMERATION OF
    (USER_DEFINED_COST,
     PERFORMANCE,
     SETUP,
     ACCESS,
     REENTRY);
  END_TYPE;
( *

```

5.1.9 diagnosis_value

This defines a type that enables specification of the type of outcome associated with a diagnosis. For a particular diagnosis, the set of outcomes shall include, at a minimum, GOOD and CANDIDATE. Further, no more than one of each is permitted.

EXPRESS specification:

```

*)
  TYPE diagnosis_value = ENUMERATION OF
    (GOOD,
     CANDIDATE,
     USER_DEFINED_DIAGNOSIS_OUTCOME,
     DIAGNOSIS_NOT_KNOWN);
  END_TYPE;
( *

```

5.1.10 goal

Type goal defines an enumeration of possible optimization criteria used to guide diagnosis. Example criteria include minimum cost, minimum expected time, and minimum number of tests.

EXPRESS specification:

```
*)
  TYPE goal = ENUMERATION OF
    (MINIMUM_TESTS,
     MINIMUM_NON_TIME_COST,
     MINIMUM_TIME_COST,
     MINIMUM_EXPECTED_NON_TIME_COST,
     MINIMUM_EXPECTED_TIME_COST,
     USER_DEFINED_OPTIMIZATION);
  END_TYPE;
( *
```

5.1.11 non_time_unit

This type defines legal units for non-time-related costs. Non-time costs are typically incurred in terms of monetary amounts; however, the user-defined value allows for other types of non-time costs to be included.

EXPRESS specification:

```
*)
  TYPE non_time_unit = ENUMERATION OF
    (USER_DEFINED_NON_TIME,
     U_S_DOLLAR,
     POUND_STERLING,
     YEN,
     EURO,
     COUNT);
  END_TYPE;
( *
```

5.1.12 role

This type defines the role associated with a purpose in the required context of a hierarchical element.

EXPRESS specification:

```
*)
  TYPE role = ENUMERATION OF
    (TRAINING,
     VERIFICATION,
     SCHEDULED_MAINTENANCE,
     MAINTENANCE_ACTION,
     READY_FOR_ISSUE,
     USER_DEFINED_ROLE);
  END_TYPE;
( *
```

5.1.13 severity_category

Enumerated type `severity_category` is used to assign one of four standard values to the severity attribute. Values assigned are one of CATASTROPHIC, CRITICAL, MARGINAL, or MINOR (in decreasing order of severity). Note that this can be used in conjunction with failure probability information (derived from failure rate) to determine the criticality of a diagnosis.

EXPRESS specification:

```
*)
  TYPE severity_category = ENUMERATION OF
    (CATASTROPHIC,
     CRITICAL,
     MARGINAL,
     MINOR) ;
  END_TYPE;
( *
```

5.1.14 test_value

Type `test_value` is used to characterize outcomes that are associated with a test.

EXPRESS specification:

```
*)
  TYPE test_value = ENUMERATION OF
    (PASS,
     FAIL,
     TEST_NOT_KNOWN,
     TEST_NOT_AVAILABLE,
     USER_DEFINED_TEST_OUTCOME) ;
  END_TYPE;
( *
```

5.1.15 time_basis

This type permits specification of the basis for recording time-related information. Specifically, it permits specification of operating time, clock time, or some user-defined version of time.

EXPRESS specification:

```
*)
  TYPE time_basis = ENUMERATION OF
    (NOT_KNOWN,
     OPERATING_TIME,
     CLOCK_TIME,
     USER_DEFINED_BASIS) ;
  END_TYPE;
( *
```

5.1.16 time_unit

This type defines legal units for time-related costs. The most common units of time (as derived from IEEE Std 716TM-1995 [B9], which is referred to as the ATLAS standard) are specified; however, the user-defined value allows for other types of time costs to be included.

EXPRESS specification:

```
*)
  TYPE time_unit = ENUMERATION OF
    (USER_DEFINED_TIME,
     HOURS,
     MINUTES,
     SECONDS,
     MSEC,
     USEC,
     NSEC,
     PSEC);
  END_TYPE;
(*
```

5.1.17 action

Entity action is a specific action taken either to test or repair an item. Actions are the primary entities against which costs are associated. Ultimately, the diagnosis and repair processes are concerned with optimizing the sequence of actions required to return a unit to service.

EXPRESS specification:

```
*)
  ENTITY action;
    has_name          : name_type;
    has_description    : description_type;
    has_cost           : OPTIONAL SET [1:?] OF cost;
    required_resource  : OPTIONAL SET [1:?] OF resource;
    action_frequency   : OPTIONAL frequency;
    order_constraint    : OPTIONAL order_operator;
  UNIQUE
    one_name : has_name;
  END_ENTITY;
(*
```

Attribute definitions:

has_name	: Provides a unique name for the action.
has_description	: Provides a description of the action.
has_cost	: A set of costs associated with the action represented by this entity; the set is optional but, if specified, shall include at least one cost attribute.
required_resource	: Identifies the resources required to perform a given action. This attribute is optional with a minimum cardinality of one, should it exist.

`action_frequency` : Specifies, for periodic actions, how frequently a given action needs to be performed.

`order_constraint` : Attribute `order_constraint` identifies an ordering precondition that must be satisfied for a particular action to be performed.

5.1.18 atom

Entity `atom` is a primitive element within the first-order logic. An atom can be either a constant (e.g., TRUE, FALSE, or some number) or a variable that takes on some value. This is a subtype of `logic_element` and a supertype of one of `variable` or `const`.

EXPRESS specification:

```
* )
  ENTITY atom
    ABSTRACT SUPERTYPE OF (ONEOF(variable, const))
    SUBTYPE OF(logic_element);
  END_ENTITY;
( *
```

5.1.19 const

A constant within the first-order logic. This is a subtype of `atom` and has an associated value.

EXPRESS specification:

```
* )
  ENTITY const
    SUBTYPE OF(atom);
    constant_value      : name_type;
  END_ENTITY;
( *
```

Attribute definitions:

`constant_value` : Specifies the particular value to be taken on by a constant.

5.1.20 cost

Costs are categorized by the type of cost to which they relate. One dimension of the cost set identifies whether the cost is a measure of time in seconds (or some other unit), or if it is a calculated cost. A time-related cost is a measure of the time it takes to perform a task. A non-time-related cost is an expense that is computed, perhaps in financial terms or by an objective function. The second dimension to the cost group is based on the task to which the cost pertains: performance, setup, access, re-entry.

Each cost has an associated unit to enable consistent processing of included values. Two types of units are defined as enumerated types and include `time_unit` and `non_time_unit`. `time_unit` is an enumerated type that defines the time units to be associated with a time-related cost. `non_time_unit`

is an enumerated type covering predominant international currencies and defines types of cost units to be associated with `non_time_related_costs`.

EXPRESS specification:

```

*)
ENTITY cost
  ABSTRACT SUPERTYPE OF (ONEOF(time_cost, non_time_cost));
  cost_element          : OPTIONAL cost_type;
  upper                 : OPTIONAL bound;
  lower                 : OPTIONAL bound;
  predicted_value       : OPTIONAL cost_value;
  criterion              : SET [1:?] OF cost_category;
WHERE
  valid_bound : NOT(EXISTS(upper)) OR
               NOT(EXISTS(lower)) OR
               (lower <= upper);
  valid_upper_bound : NOT(EXISTS(predicted_value)) OR
                     NOT(EXISTS(upper)) OR
                     (predicted_value <= upper);
  valid_lower_bound : NOT(EXISTS(predicted_value)) OR
                     NOT(EXISTS(lower)) OR
                     (lower <= predicted_value);
END_ENTITY;
( *

```

Attribute definitions:

`cost_element` : An attribute that specifies what type of cost is being defined. The type is an enumeration consisting of `user_defined`, `performance`, `setup`, `access`, and `re-entry`. One of these cost types shall be specified.

`upper` : An optional attribute providing the nominal upper bound for the value of the cost entity.

`lower` : An optional attribute providing the nominal lower bound for the value of the cost entity.

`predicted_value` : An attribute providing the nominal, expected, or some other predicted value for the cost entity.

`criterion` : Attribute `criterion` identifies the cost criterion that will be used to classify the cost parameter. For example, `criterion` can identify skill level as a non-time cost. This attribute is a set because a particular cost item can represent multiple criteria.

Formal propositions:

`valid_bound` : When they exist, the lower and upper attributes are constrained to values such that the value of the lower attribute is less than or equal to the upper attribute. This constraint permits only one of the bounds to exist.

`valid_upper_bound` : When they exist, the `expected_value` and upper attributes are constrained to values such that the value of the `expected_value` attribute is less than or equal to the upper attribute.

`valid_lower_bound` : When they exist, the `expected_value` and lower attributes are constrained to values such that the value of the lower attribute is less than or equal to the `expected_value` attribute.

5.1.21 cost_categoryEXPRESS specification:

```

*)
  ENTITY cost_category;
    description      : description_type;
    name             : name_type;
  UNIQUE
    one_name         : name;
  END_ENTITY;
( *

```

Attribute definitions:

description : Attribute description is used to provide an elaborated description of what the cost category is.

name : Attribute name provides a means for identifying the cost category.

5.1.22 cost_rate

In addition to providing basic time and cost units, it may be necessary to associate a `cost_rate` with an entity (e.g., resource). The `cost_rate` attribute defines a ratio of non-time cost to some unit of time and can be used, e.g., to specify labor rates or rental rates.

EXPRESS specification:

```

*)
  ENTITY cost_rate;
    time_cost_unit      : time_unit;
    denominator         : cost_value;
  WHERE
    denominator_is_non_zero : SELF.denominator <> 0;
  END_ENTITY;
( *

```

Attribute definitions:

time_cost_unit : An attribute to provide the time units for computing the cost rate. In other words, this attribute will provide the time part of cost per unit time.

denominator : An attribute that provides a divisor for the `cost_rate`, thus permitting values other than unity.

Formal propositions:

denominator_is_non_zero : Since the `cost_rate` entity defines a ratio that permits specification of the denominator, this rule constrains the denominator to be nonzero, thus preventing dividing by zero.

5.1.23 diagnosis

Diagnosis can be used for the representation of a lattice of diagnostic conclusions. Typically, a diagnosis consists of failures or fault. The diagnoses can be interconnected in a lattice to classify constituent units, e.g., by function. This construct provides a generalized grouping mechanism for the constituents of the unit under test and their failure modes. The diagnosis lattice resulting from using a structure of these entities shall not be cyclic, in the sense that no diagnosis be an ancestor of itself.

Example: A diagnosis might be of system under test #4 (flight computer), consisting of a diagnosis of BOARD #1 (A/D converter), BOARD #2 (CPU), BOARD #3 (Display), etc. A diagnosis of BOARD #3 may consist of diagnoses of CHIP #1, CHIP #2, CHIP #3, and all may be at different levels.

EXPRESS specification:

```

*)
ENTITY diagnosis
  SUPERTYPE OF (ONEOF(failure, fault))
  SUBTYPE OF (hierarchical_element);
  has_outcome          : OPTIONAL SET [2:?] OF diagnosis_outcome;
  has_rate              : OPTIONAL failure_rate;
  mechanism            : OPTIONAL SET [1:?] OF failure_mode;
  severity              : OPTIONAL severity_category;
WHERE
  outcomes_required_for_atomic_diagnosis : (SIZEOF(SELF.members) >
    0) OR EXISTS(has_outcome);
  minimal_outcomes : (NOT(EXISTS(has_outcome))) XOR
    ((SIZEOF(QUERY(tmp <* has_outcome
      tmp.standard_diagnosis_value = GOOD)) = 1) AND
    (SIZEOF(QUERY(tmp <* has_outcome
      tmp.standard_diagnosis_value = CANDIDATE)) = 1));
  mechanism_at_leaves : NOT(EXISTS(mechanism)) OR (SIZEOF(members) = 0);
  faults_at_leaves : (NOT('AI_ESTATE_COMMON_ELEMENT_MODEL.fault' IN
    TYPEOF(SELF))) OR
    (SIZEOF(members) = 0);
  children_same_type : QUERY(tmp <* SELF.members
    NOT('AI_ESTATE_COMMON_ELEMENT_MODEL.diagnosis'
    IN TYPEOF(tmp))) = [];
END_ENTITY;
( *

```

Attribute definitions:

has_outcome : A set of two or more outcomes that are the expected outcomes of the diagnosis. An outcome is a characterization of the logical value for the corresponding truth or falsity of the diagnosis applying to the current state. This attribute is shown to be optional; however, it is constrained such that it is required if the diagnosis has no member diagnoses. Since diagnosis outcomes are the basis for determining the final conclusions in AI-ESTATE, outcomes shall be available.

NOTE—Diagnoses in the data model are not restricted to two outcomes; any number of outcomes are permitted for a test, but at least two outcomes are required.

<code>has_rate</code>	: A value indicating the measure of the expected failure rate of the diagnosis, i.e., the number of failures per time span. This value can be converted to a probability of failure.
<code>mechanism</code>	: Identifies the failure mode or modes associated with this diagnosis.
<code>severity</code>	: Attribute <code>severity</code> associates a level of severity for the given diagnosis. This attribute is optional since not all models need to use this information. Criticality can be derived by selecting all of the diagnoses at a particular severity level and then ranking them by their failure probabilities, which are derived from the failure rates.

Formal propositions:

<code>outcomes_required_for_atomic_diagnosis</code>	: Determines whether or not outcomes are associated with a diagnosis and requires that an atomic diagnosis (i.e., a diagnosis for which there are no sub-diagnoses) have outcomes. The cardinality on the <code>diagnosis_outcome</code> set ensures that, should outcomes exist, there are at least two of them. Note that nonatomic diagnoses are permitted, but not required, to have diagnostic outcomes.
<code>minimal_outcomes</code>	: Requires that either the set of outcomes not be defined, or that the set of outcomes include, at a minimum, exactly one outcome of value GOOD and exactly one outcome of value CANDIDATE.
<code>mechanism_at_leaves</code>	: This rule ensures that a mechanism (i.e., a cause) for a given diagnosis to be true only occurs at the leaf of the diagnosis lattice. Note that the leaf is not required to provide the mechanism.
<code>faults_at_leaves</code>	: This rule ensures that a diagnosis of type <code>fault</code> can only exist at the leaf of the diagnosis lattice. Note that the leaf of the lattice is not required to be of type <code>fault</code> .
<code>children_same_type</code>	: Ensures that all children (i.e., members) of a diagnosis are also of type diagnosis.

5.1.24 diagnosis_outcome

Entity `diagnosis_outcome` is a set of outcomes associated with any diagnosis in a model. Diagnosis outcomes associate discrete values to ranges of states in the diagnosis definition and form the basis for the diagnostic process. For any given diagnosis in an AI-ESTATE model, at least two outcomes shall be defined. Thus, at a minimum, outcomes corresponding, e.g., to good/candidate should be provided with a given specification. At times more than two outcomes may be appropriate, e.g., good/suspect/candidate/bad.

EXPRESS specification:

```
*)
  ENTITY diagnosis_outcome
    SUBTYPE OF (outcome);
    standard_diagnosis_value      : diagnosis_value;
    INVERSE
      for_diagnosis                : diagnosis FOR has_outcome;
  END_ENTITY;
(*
```

Attribute definitions:

standard_diagnosis_value : This attribute identifies whether the outcome is a standard value from the perspective of traditional diagnostic outcomes. In most model-based systems, the standard values are GOOD and CANDIDATE. A third value, USER_DEFINED_DIAGNOSIS_OUTCOME, is used to indicate whether additional outcomes have been defined.

for_diagnosis : Links the outcome to the specific diagnosis with that outcome.

5.1.25 diagnostic_model

Diagnostic reasoning involves drawing conclusions from test outcomes. The relationships between test outcomes and diagnostic conclusions are defined in diagnostic models. A diagnostic model is a collection of entities that provide information about the system under test. A `diagnostic_model` refers to some AI-ESTATE conformant model as specified in this standard. Entities from the Common Element Model are referenced as required by those diagnostic models.

EXPRESS specification:

```
*)
  ENTITY diagnostic_model;
    name                : name_type;
    description          : description_type;
    model_element        : SET [3:?] OF hierarchical_element;
    orders_steps         : OPTIONAL SET [1:?] OF order_operator;
  UNIQUE
    one_name : name;
  WHERE
    one_diagnosis_is_no_fault : (SIZEOF(QUERY(tmp <* model_element |
                                          tmp.name = 'no_fault')) = 1);
    element_is_rollup : element_rollup(SELF,SELF.model_element);
  END_ENTITY;
(*
```

Attribute definitions:

name	: An attribute used to identify the entity uniquely.
description	: Attribute used to provide an elaborated explanation of the model.
model_element	: An attribute to identify the various model elements in a diagnostic model. Model elements are any of the hierarchical elements as defined by the information model (i.e., tests, diagnoses, repairs, resources, functions).
orders_steps	: This attribute identifies an optional set of ordering operators to override any optimization process that may be applied for diagnosis.

Formal propositions:

one_diagnosis_is_no_fault	: This rule verifies that one of the diagnosis entities defined in the model has a name that corresponds to the constant no_fault. This rule ensures that any AI-ESTATE conformant model explicitly defines a special diagnostic conclusion indicating no fault found.
element_is_rollup	: This rule verifies that all of the hierarchical elements referenced at this level are defined as being part of this model. There is a companion rule for the hierarchical elements to ensure that they are listed in the rollup at this level.

5.1.26 failure

A failure is a manifestation of a fault within a system. When considering a functional model, it is the failure of the system under test to perform some intended function.

EXPRESS specification:

```

*)
  ENTITY failure
    SUBTYPE OF (diagnosis);
    failed_item    : OPTIONAL func;
  END_ENTITY;
(*

```

Attribute definitions:

failed_item	: Identifies the specific element (i.e., function) that has failed. This is a direct tie to the functional part of the system associated with a diagnosis.
-------------	--

5.1.27 failure_mode

A failure mode is a specific mode or means by which a unit or system can fail. The specific failure mode is associated, first with the diagnosis, then (by way of the appropriate subtype) to the item that has failed.

EXPRESS specification:

```
*)
  ENTITY failure_mode;
    name                : name_type;
    description          : description_type;
  UNIQUE
    one_name   : name;
  END_ENTITY;
( *
```

Attribute definitions:

name : Used to identify the failure mode uniquely.
description : Attribute used to provide an elaborated explanation of what the failure mode is.

5.1.28 failure_rate

This entity indicates the rate of occurrence of a diagnosis (including its subtypes fault and failure). The failure_rate can either be an actual observed value or an estimate. To specify failure_rate, a time_basis shall be defined. Time_basis is used in the context of determining the computational basis for failure rate statistics.

EXPRESS specification:

```
*)
  ENTITY failure_rate;
    description : description_type;
    basis       : time_basis;
  END_ENTITY;
( *
```

Attribute definitions:

description : Attribute used to provide an elaborated explanation of the failure rate.
basis : An attribute to define the basis for computing the failure rate. In particular, this attribute identifies if the failure_rate is computed based on operation time or clock time.

5.1.29 fault

A fault is the physical cause of anomalous behavior within a system. A fault typically corresponds to some physical breakdown in the system under test.

EXPRESS specification:

```
*)
  ENTITY fault
    SUBTYPE OF (diagnosis);
    failed_item : OPTIONAL repair_item;
  END_ENTITY;
( *
```

Attribute definitions:

`failed_item` : Identifies the specific element (i.e., repair item) that has failed. This is a direct tie to the physical part of the system associated with a diagnosis.

5.1.30 frequency

Frequency is the rate (i.e., the number of times something occurs per unit time).

EXPRESS specification:

```
*)
  ENTITY frequency;
    time_span      : time_cost;
    action_count   : non_negative_integer;
  END_ENTITY;
(*
```

Attribute definitions:

`time_span` : Attribute `time_span` identifies the span of time between periodic actions. While this attribute is of type `time_cost` note that the semantics of this type are identical to a time-related entity.

`action_count` : Attribute `action_count` identifies the number of times in a time span the action is performed.

5.1.31 func

A function corresponds to any “functional behavior” represented within a system. The `func` entity is intended to provide a placeholder for functional information in the event the diagnostic or testability model is function-oriented. Note the type of entity cannot be “function” since this is a reserved word in EXPRESS.

EXPRESS specification:

```
*)
  ENTITY func
    SUBTYPE OF (hierarchical_element);

  INVERSE
    implemented_by      : SET [1:?] OF repair_item FOR includes_function;
  WHERE
    children_same_type  : QUERY (tmp <* SELF.members |
                                NOT ('AI_ESTATE_COMMON_ELEMENT_MODEL.func'
                                     IN TYPEOF (tmp))) = [] ;
  END_ENTITY;
(*
```

Attribute definitions:

`implemented_by` : This attribute identifies the repair item or items used to implement a particular function within a system.