

IEEE Std 1232.1-1997

IEEE Std 1232.1-1997

IEEE Trial-Use Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE): Data and Knowledge Specification

**IEEE Standards Coordinating Committee 20 (SCC 20) on
Abbreviated Test Language for All Systems (ATLAS)**

5 September 1997

SH 94521

IEEE Trial-Use Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE): Data and Knowledge Specification

Sponsored by

**IEEE Standards Coordinating Committee 20 (SCC 20) on
Abbreviated Test Language for All Systems (ATLAS)**

Approved 20 March 1997

IEEE Standards Board

Abstract: Formal models for information used in system diagnosis are defined. As part of the AI-ESTATE set of standards, this standard includes several models that form the basis for a format to facilitate exchange of persistent diagnostic information between two reasoners, and also provides a formal typing system for the services defined in the AI-ESTATE service specification.

Keywords: AI-ESTATE, artificial intelligence, diagnosis, enhanced diagnostic inference model, fault tree model

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1997 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1997. Printed in the United States of America.

ISBN 1-55937-912-X

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 Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. 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.

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.

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 all 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 on standards and requests for interpretations should be addressed to:

Secretary, IEEE 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; (508) 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 1232.1-1997, IEEE Trial-Use Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE): Data and Knowledge Specification.]

The AI-ESTATE (Artificial Intelligence Exchange and Service Tie to All Test Environments) set of standards is intended to provide 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 being defined whereby diagnostic tools can be developed to process the diagnostic information in a consistent and reliable way.

This component standard of the AI-ESTATE family defines formal models for information used in system diagnosis. It is not the intent of IEEE Std 1232.1-1997 to define the information needed for testing systems. Such standards exist (e.g., IEEE Std 716-1995, IEEE Std 1226-1993, and IEEE Std 1149.1-1990) and can be used in conjunction with this standard. This standard defines the relationships between test information and the diagnoses that can be drawn from that information.

Specifically, this standard includes several models, defined in EXPRESS (ISO 10303-11: 1994), which are used in two ways. First, these models form the basis for a format to facilitate exchange of persistent diagnostic information between two reasoners. The format used is based on EXPRESS-I (ISO/TR 10303-12: 1997) and is directly derivable from the EXPRESS models. Second, these models provide a formal typing system for the services defined in the AI-ESTATE service specification. The types of information used by the services defined by AI-ESTATE shall conform to the information defined by the models in this standard.

This standard is organized with a model in the body defining common entities to all types of diagnosis. Several normative annexes are included that define reasoning-specific models. Currently, normative annexes have been defined for a fault tree model (i.e., a fixed decision tree based on identified test outcomes) and an enhanced diagnostic inference model (i.e., a propositional model tying test outcomes to possible diagnoses). Both types of models are in wide use in industry today, which indicates their wide acceptance in system diagnoses. It is anticipated that additional models will be defined in the future, perhaps addressing concerns of diagnostic constraints, diagnostic rules, and diagnostic neural networks. In all cases, the underlying theme will be diagnosis.

Publication of this trial-use standard for comment and criticism has been approved by the Institute of Electrical and Electronics Engineers. Trial-use standards are effective for 24 months from the date of publication. Comments for revision will be accepted for 18 months after publication. Suggestions for revision should be directed to the Secretary, IEEE Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, and should be received no later than 5 September 1999. It is expected that following the 24-month period, this trial-use standard, revised as necessary, shall be submitted to the IEEE Standards Board for approval as a full-use standard.

At the time that this standard was completed, the AI-ESTATE subcommittee had the following membership:

John W. Sheppard, *Data and Knowledge Representation Working Group, Co-Chair*
Gregory P. Bowman, *Data and Knowledge Representation Working Group, Co-Chair*
Leslie A. Orlidge, *AI-ESTATE, Co-Chair*
Helmut Scheibenzuber, *AI-ESTATE, Co-Chair*

Guy Adam	Greg Griffes	John Powell
Darrell Bartz	Gary Hardenburg	N. Ramachandran
Michael Blair	Mike Heilman	Bill Reeves
Gregory Bowman	John Heiser	Bill Rodriguez
David Brooks	Francis Kearns	David Roggendorff
Lawrence D. Carpenter	James Lewis	Howard Savage
Elaine Cashar	Richard Maguire	Mike Seavey
Richard Churcher	Michael Martin	John Sheppard
George Clarius	Robert McGarvey	Lee Shombert
Christopher Connelly	Harry McGuckin	William R Simpson
James Cottrell	James Mills	Joseph Stanco
Bernard Dathy	Mukund Modi	William Taylor
Somnath Deb	Tom Newton	Kirk Thompson
Bernd Dinkluge	Dave Nichols	Jörg Urban
Rob Fletcher	Hans Obermeier	Richard Weger
Steve Fortier	Ed Perkins	Dan Weiss
Ken Fox	Jean Pouilly	Tim Wilmering
Arnold Greenspan		Will Young

Other individuals who have contributed review and comments are:

Jonas Åstrand	Jacques Flandrois	Brian Kelle
Timothy Bears	Sharon Goodall	Paul Knese
Christus Bezirtzoglou	T. Winston Griffin Arthur Hann	Mats Landerholm
Maria Bostic	James Hanna	Richard Lazarus
Patrice Brard	Leonard Haynes	Jiunn-Jyh Lin
Ronald Brooks	Dan Inman	Peter Lord
Frank Conforti	Jon J. Kasprick	Colin Maunder
Serhat Enan	Helmuth Kaunzinge	Krishna Pattipati
Christine Fisher		Richard Walker

The following persons were on the balloting committee:

Guy Adam
Michel Bantegnie
Jean-Yves Barbier
Darrell O. Bartz
Stephen M. Bowen
Gregory Bowman
Alan L. Bridges
Antonius Bunsen
Larry D. Carpenter
E. Elaine Cashar
Richard D. Churcher
William Colmer
Bernard Dathy
Jeffrey S. Dean
Bernd Dinklage
Michael T. Ellis
Horst D. Ey
William Brit Frank
Sharon Goodall
Michael N. Granieri
Gary L. Hardenburg
Raymond L. Hasenstab

Robert G. Hayes
Mike Heilman
John E. Heiser
Wesley Higaki
John Leuthe
James L. Lewis
Alfred Lowenstein
Michael S. Martin
Robert L. McGarvey
James M. Mills
Mukund U. Modi
Charles Morris
David S. Morris
Richard E. Neese
Hans Obermeier
Roy Oishi
Leslie A. Orlidge
Dennis C. Peterson
Narayanan Ramachandran
William E. Reeves
David Roggendorff
William E. Russell, Jr.
Eric Sacher

Helmut Scheibenzuber
John B. Schroeder
Michael L. Seavey
John W. Sheppard
Lee A. Shombert
William R. Simpson
Joseph J. Stanco
Robert Stasonis
Lawrence J. Stockwell
William John Taylor
Kirk D. Thompson
C. Richard Unkle
Ronald Waxman
J. Richard Weger
Daniel H. Weiss
Lynn Wheelwright
Kenneth D. Wilkinson
Roger L. Williams
Timothy J. Wilmering
Gregory Winter
Will W. Young
Robert W. Zimbardi

When the IEEE Standards Board approved this standard on 20 March 1997, it had the following membership:

Donald C. Loughry, *Chair*

Richard J. Holleman, *Vice Chair*

Andrew G. Salem, *Secretary*

Clyde R. Camp
Stephen L. Diamond
Harold E. Epstein
Donald C. Fleckenstein
Jay Forster*
Thomas F. Garrity
Donald N. Heirman
Jim Isaak
Ben C. Johnson

Lowell Johnson
Robert Kennelly
E. G. "Al" Kiener
Joseph L. Koepfinger*
Stephen R. Lambert
Lawrence V. McCall
L. Bruce McClung
Marco W. Migliaro

Louis-François Pau
Gerald H. Peterson
John W. Pope
Jose R. Ramos
Ronald H. Reimer
Ingo Rüschi
John S. Ryan
Chee Kiow Tan
Howard L. Wolfman

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
Alan H. Cookson

Lisa S. Young
IEEE Standards Project Editor

Contents

CLAUSE	PAGE
1. Overview.....	1
1.1 Scope.....	2
1.2 Purpose.....	3
1.3 Application.....	3
1.4 Conventions used in this document	4
2. References.....	4
3. Definitions.....	5
4. Description.....	5
4.1 Objectives	5
4.2 Interchange format	6
4.3 Relationship to AI-ESTATE architecture	6
4.4 Extensibility	6
5. Specification	8
5.1 Overview.....	8
5.2 AI-ESTATE Common Element Model	8
6. Conformance.....	46
ANNEX	
Annex A (informative) Bibliography	47
Annex B (informative) Summary of EXPRESS Terms	50
B.1 Schema	50
B.2 Entity	51
B.3 Attribute	51
B.4 Type Definition.....	52
B.5 Subtypes/Supertypes	53
B.6 External schema references.....	54
B.7 Constraints and WHERE clauses.....	55
B.8 Page references in EXPRESS-G	56
B.9 Functions.....	57

ANNEX	PAGE
Annex C (normative) AI-ESTATE Fault Tree Model	58
C.1 fault_tree_model	59
C.2 fault_tree_step.....	60
C.3 test_result	61
C.4 Fault Tree Model EXPRESS specification	62
C.5 Example model	63
C.6 Conformance.....	77
Annex D (normative) AI-ESTATE Enhanced Diagnostic Inference Model.....	79
D.1 diagnostic_inference	80
D.2 enhanced_diagnostic_inference_model	81
D.3 inference.....	81
D.4 outcome_inference.....	83
D.5 test_inference	84
D.6 Enhanced diagnostic inference model EXPRESS specification	84
D.7 Example model	85
D.8 Conformance.....	102

IEEE Trial-Use Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE): Data and Knowledge Specification

1. Overview

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.

This standard is one component of a set of AI-ESTATE standards. The data and knowledge specification provides a standard representation of the common data elements required for system test and diagnosis. This will ensure portability of test-related knowledge bases for intelligent system test and diagnosis.

The goals of this component of the AI-ESTATE standard, formally defined in IEEE Std 1232-1995,¹ are summarized as follows:

- Incorporate domain specific terminology;
- Facilitate portability of diagnostic knowledge;
- Permit extensibility of diagnostic knowledge.

This standard shall use the vocabulary and semantics of the test and diagnosis domain by incorporating terminology from Clause 3 of IEEE Std 1232-1995, existing IEEE standards, and trial-use standards where possible. Deviations from this shall be kept to a minimum and shall be explicitly defined to the user at first occurrence.

This AI-ESTATE standard shall provide a controlled extension mechanism to allow inclusion of new technology outside the scope of the AI-ESTATE data and knowledge specification definitions.

IEEE Std 1232.1-1997 defines 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

¹Information on references can be found in Clause 2.

shall support fully portable diagnostic knowledge. No host computer dependence shall be contained in the AI-ESTATE standards. Host system dependence shall not be included in the AI-ESTATE data and knowledge specification.

IEEE Std 1232.1-1997 shall define key data and knowledge specification formats. Implementations that use only these specification formats will be portable. This shall not preclude use of AI-ESTATE interfaces with nonconformant specification formats; however, such implementations may not be portable.

Specifications will insure the portability of data and knowledge among AI-ESTATE implementations through interchange formats. This will eliminate the respecification of data and knowledge in moving from one AI-ESTATE implementation to another and will maximize the reuse and sharing of maintenance expert knowledge. As shown in Figure 1, information can be moved from one 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. The translation step is not a requirement; an AI-ESTATE implementation may use the interchange format for its own internal form.

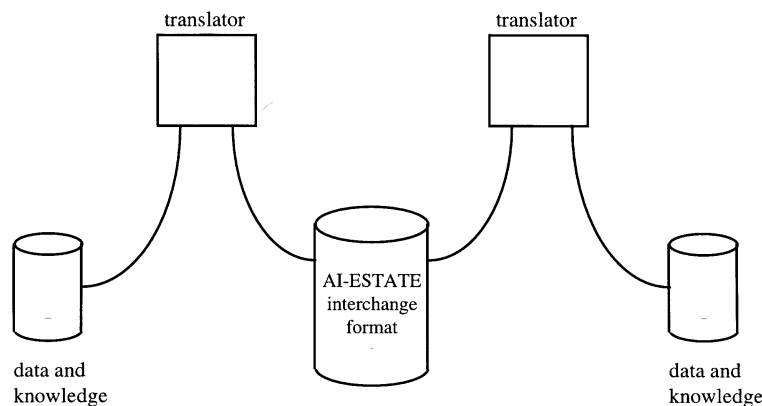


Figure 1—An example of AI-ESTATE's portability mechanism for data and knowledge

Traceability of data and knowledge within AI-ESTATE applications is important. This standard relies on other standards to satisfy the traceability requirements, but does not specify which of these standards shall be used. Two possible standards that could be used are ISO 10303-41: 1994 [B24] and ISO 10303-44: 1994 [B25].²

1.1 Scope

This standard defines the form of the data and knowledge needed to support AI-ESTATE. The constructs defined in the body of this standard are available for use by the normative annexes of this standard of AI-ESTATE, which provide formal data models oriented towards specific approaches to diagnostic reasoning. The constructs defined here are applicable to the domain of equipment test and diagnosis, independent of the reasoning approach.

²The numbers in brackets preceded by the letter B correspond to those of the bibliography in Annex A.

1.2 Purpose

IEEE Std 1232.1-1997 defines information models for knowledge bases that are to be used in the context of test and diagnosis, and from these models, to derive a data interchange format. This standard provides the formal specification of the data and knowledge necessary to perform system test and diagnosis and insures the portability of data and knowledge in AI-ESTATE-conformant systems. This standard specifies the requirements for the machine readable form of the data needed to support all components of the AI-ESTATE set of standards.

1.3 Application

This standard facilitates the use of standard representations of diagnostic data and knowledge within the context of an AI-ESTATE implementation. For example, data and knowledge specifications for Enhanced Diagnostic Inference Models and Fault Tree Models have been provided as normative annexes.

In specifying data and knowledge for these domains, a structure has been constructed, as shown in Figure 2. 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 test subject), **anomaly** (failure modes of the test subject), **resource**, and **test**. These constructs are characterized by attributes such as costs and failure rates that are also specified in the Common Element Model.

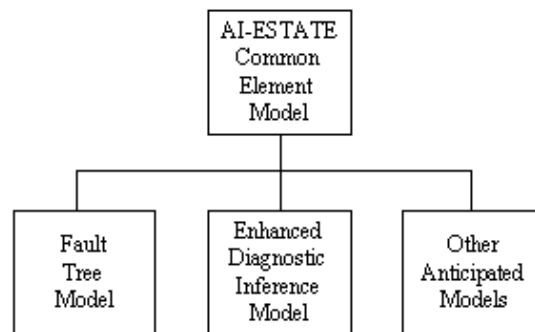


Figure 2—Hierarchical structure of AI-ESTATE models

Below this top layer is a layer of application-specific data and knowledge formats (e.g., 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 requirements.

AI-ESTATE provides data and knowledge specifications for an Enhanced Diagnostic Inference Model. Enhanced diagnostic inference models consist of elements that represent devices, their parts, the failure modes of those parts or any logical diagnostic conclusion, and the tests of those devices. Central to an enhanced diagnostic inference model are the inferences or dependencies of the test subject and its tests. AI-ESTATE provides the formal specification of the data and knowledge necessary to perform system test and diagnosis with an enhanced diagnostic inference model based reasoner and will insure the portability of this information among AI-ESTATE conformant reasoner systems.

AI-ESTATE provides data and knowledge specifications for a Fault Tree Model. The Fault Tree Model is a representation of a decision tree with fixed fault isolation strategies. It is constructed *a priori* and remains static during the diagnosis. Although not generally classified as an AI technique, this specification of the fault tree is included, since it is frequently used as the primary diagnostic strategy or in conjunction with other test generation strategies.

Other data and knowledge specification formats are envisioned and will be included in revisions of this standard. Examples include a constraint-based model, a neural network model, and formats to support the other components of the AI-ESTATE standard. Models supporting these are shown in Figure 2 as “Other Anticipated Models.” The Common Element Model has been specified such that other data and knowledge specification formats can utilize its constructs as base elements that are tailored to the particular application’s needs.

1.4 Conventions used in this document

Clause 5 of IEEE Std 1232.1-1997 presents the formal specification for the data and knowledge of this standard. The subclauses present specification formats using the EXPRESS language and use the following conventions in their presentation:

- a) 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.
- b) Each entity of each EXPRESS schema is presented in a separate subclause. The subclause structure begins with the actual EXPRESS specification; then each attribute of the entity is described below the Attribute Definition heading. If constraints have been specified, these are described below the Formal Propositions heading.
- c) A component standard of IEEE Std 1232-1995 may define an EXPRESS schema that contains subtypes of entities that are declared in other schemata of IEEE Std 1232-1995. Unless otherwise stated, the supertype entity contains an implicit SUPERTYPE ONEOF clause that names the supertype entity.
- d) NOTE—This mechanism is not meant to replace proper EXPRESS coding. All such implicit SUPERTYPE clauses should be made explicit at the next opportunity in the standardization cycle.
- e) This standard uses the vocabulary and definitions of relevant IEEE standards. In case of a conflict of definitions, the following precedence shall be observed: 1) AI-ESTATE definitions found in Clause 3; 2) SCC20 documentation and standards; and 3) IEEE Std 100-1996.

2. References

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

IEEE Std 100-1996, The IEEE Standard Dictionary of Electrical and Electronics Terms (ANSI).³

IEEE Std 1232-1995, Trial-Use Standard for Artificial Intelligence and Expert System Tie to Automatic Test Equipment (AI-ESTATE): Overview and Architecture

ISO 10303-11:1994, Industrial automation systems and integration—Product data representation and Exchange—Part 11: Description methods: The EXPRESS language reference manual.⁴

ISO/TR 10303-12:1997, Industrial automation systems and integration—Product data representation and exchange—Part 12: Description methods: The EXPRESS-I language reference manual.

³IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

⁴ISO publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse. 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.

3. Definitions

Clause 3 of IEEE Std 1232-1995, the base standard for the AI-ESTATE set of standards, contains a complete glossary with definitions of all the terms used in the AI-ESTATE set of standards and guides. A clear understanding of the following terms with respect to AI-ESTATE is particularly important in order to understand this standard.

3.1 element: Within AI-ESTATE, element refers to the smallest entity of a model. For example, in a particular model, the smallest test, the smallest diagnosis, and the no-fault conclusion are all elements.

3.2 encapsulation: In the context of AI-ESTATE, the act of specifying a test or collection of tests together with the associated preconditions and postconditions. Alternately, the process of hiding all of the details of an object that do not contribute to the essential characteristics.

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

3.4 fault isolation: The process of reducing the number of anomalies that comprise a diagnosis. Identification of an anomaly or anomalies to a degree sufficient to undertake an appropriate corrective action.

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

3.6 knowledge: Information structure to facilitate derivation of new information.

3.7 lattice: A partial ordering imposed on a structure of model entities.

3.8 portability: The capability of being read and/or interpreted by multiple systems.

3.9 resource allocation: The assignment of physical resources to virtual resources such that the virtual resource requirements are satisfied.

3.10 service: A software interface, frequently implemented as a software function, providing a means for communicating information between two applications.

3.11 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.

3.12 test subject: The entity to be tested. It may range from a simple to a complex system, e.g., a unit under test or a human patient.

4. Description

4.1 Objectives

Of the AI-ESTATE set of standards, IEEE Std 1232.1-1997 provides the data and knowledge specification formats required by all other components of the AI-ESTATE set of standards and by the normative annexes of this standard. The specification contained in the body of this standard is labeled the Common Element Model. Two data and knowledge specification formats have been defined in normative annexes to this standard—an enhanced diagnostic inference model and a fault tree model. Other specification formats are planned, such as those that would support constraint-based reasoning systems and neural network-based reasoning systems.

4.2 Interchange format

IEEE Std 1232.1-1997 defines 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 data interchange format will permit exchange of diagnostic knowledge bases 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 the standard defined by ISO 10303-12: 1994.

The EXPRESS-I Format was selected as the AI-ESTATE interchange format for four reasons. First, by applying an existing standard, there is no need for defining yet another standard interchange format, thus reducing the burden of preparing the standard. Second, the EXPRESS family of languages is used throughout AI-ESTATE, thus maintaining consistency in the standard. Third, the EXPRESS-I Format is human-readable, which provides a means of directly examining data files and identifying problems in translation. Finally, EXPRESS-I is a formal language that is machine-processable.

4.3 Relationship to AI-ESTATE architecture

This standard defines a set of data models for representing diagnostic data and knowledge. This knowledge is expected to be exchanged between AI-ESTATE-conformant diagnostic reasoners that reason off of the exchanged knowledge. Within the AI-ESTATE architecture, this standard satisfies requirements for data and knowledge exchange and portability between reasoners. It is anticipated that other non-AI-ESTATE standards will satisfy data exchange requirements with non-reasoner portions of a test environment.

4.4 Extensibility

It is imperative that the data and knowledge specification 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 and compliant application. This flexibility is achieved by using the EXTEND schema. In general, the extended schema will not be usable by other AI-ESTATE conformant applications. In applying the EXTEND schema, the following rules shall apply:

- f) An IEEE Std 1232.1-1997 specification may 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 the schemata, in conjunction with the schemata of IEEE Std 1232.1-1997, shall be syntactically and semantically correct. The schemata of IEEE Std 1232.1-1997 shall not be included in the extended model definition. Further, extensions shall not be used to rename or redefine data in the standard schemata.
- g) Data contained in a model definition with an EXTEND schema shall include only entity types that are declared in IEEE Std 1232.1-1992, or that are legal, previously defined extension entities, as defined in subsequent requirements.
- h) An EXTEND schema entity shall be declared to be a subtype of exactly one of either an IEEE Std 1232.1-1997 entity or a previously defined EXTEND entity schema.
- i) A model defined with EXTEND schemata shall not cause an IEEE Std 1232.1-1997 application to reach an invalid conclusion when processing the model with the extended entities removed. In other words, any conclusion drawn by an IEEE Std 1232.1-1997-conformant reasoner using an extended model without the extended entities will be consistent with the information contained in the non-extended model. Further, consistency cannot be achieved through extension.
- j) All schema, entity, type, function, and procedure identifiers declared within the EXTEND schema shall begin with the prefix `extend_`.
- k) The EXTEND schema shall use the EXPRESS reference interface specification, "REFERENCE FROM," to allow visibility of the IEEE Std 1232.1-1997 entities within the EXTEND schema.

The following is an example of EXTEND schema:

EXTEND Schema:

```
SCHEMA extend_schema_a;
REFERENCE FROM ai_estate_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;
```

5. Specification

5.1 Overview

This clause contains the specification for the Common Element Model as defined in EXPRESS. Additional models (e.g., the Fault Tree Model and the Enhanced Diagnostic Inference Model) are defined in normative annexes to this standard. A brief overview of EXPRESS and EXPRESS-G can be found in Annex B of this standard. The Schema-level Model for this specification is given in Figure 3.

EXPRESS-G View:

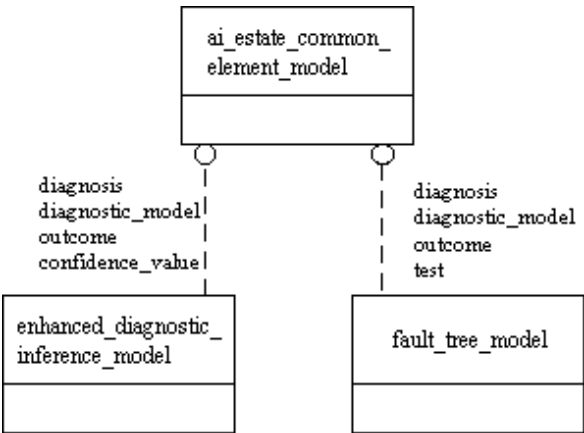


Figure 3—Complete schema-level model of AI-ESTATE data and knowledge specification

5.2 AI-ESTATE Common Element Model

The requirements of the AI-ESTATE Common Element Model schema are defined below. For each entity of the model, the EXPRESS-G view is shown, followed by the entity’s description, its EXPRESS specification, and attribute definitions. Following the entity subclauses, the EXPRESS specification is shown in its entirety. The EXPRESS-G representation of the Common Element Model is shown below. The constructs of this model are common to the AI-ESTATE domain of equipment test and diagnosis. They are intended to serve as the basic units of other data and knowledge specification formats defined for specific components of the AI-ESTATE architecture. Attributes of the model constructs, such as costs and failure rates, are also defined. Figure 4-A and 4-B illustrate the Common Element Model.

EXPRESS-G View:

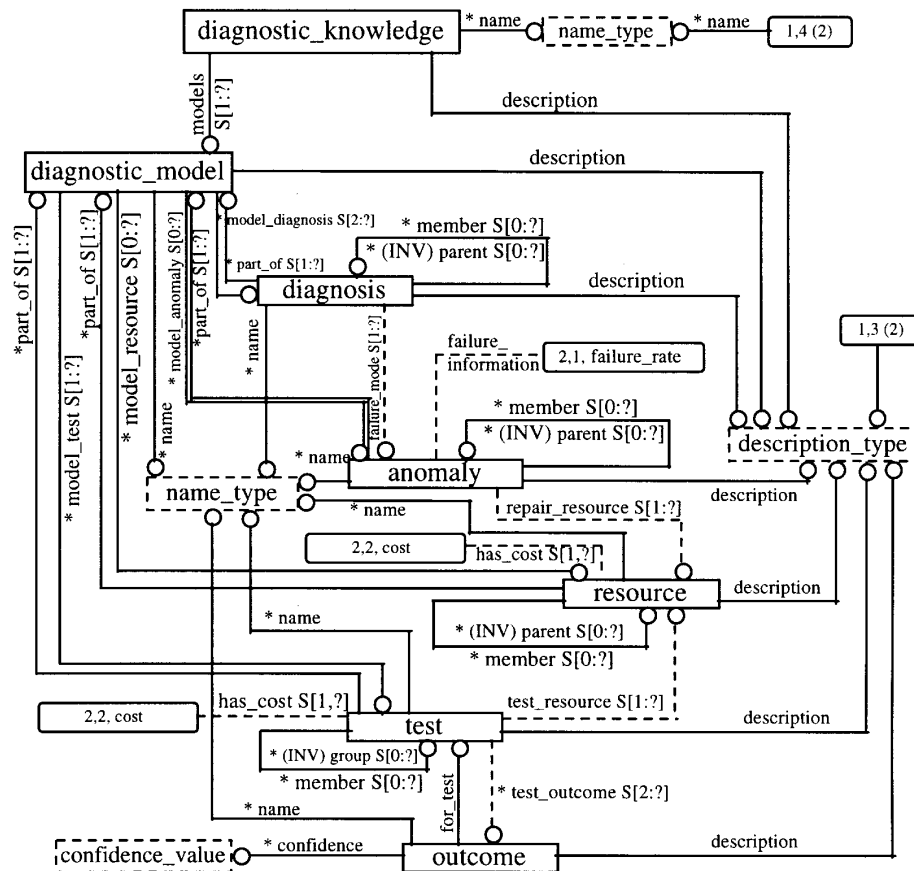


Figure 4-A—Common Element Model (diagram 1 of 2)

The Common Element Model permits the definition of the form and relationships of test subjects and tests at their most basic level. The diagnosis entity corresponds to a diagnostic conclusion about the test subject. The anomaly entity maps the fault diagnosis to a physical item in the test subject. Tests are represented as test entities in the model.

Example: Tests could be grouped by a common resource that they require. For example, a specific test suite that must be executed as a group, or a set of tests with a common setup operation might be grouped.

Example: A set of tests whose outcomes are available at the beginning of a diagnostic procedure can be specified in a model using the test entity.

The following is an alphabetical list of entities and types:

anomaly	bound	confidence_value
cost	cost_rate	cost_value
cost_type	description_type	diagnosis
diagnostic_knowledge	diagnostic_model	factor
failure_rate	hour	name_type
non_time_cost	non_time_unit	outcome
resource	test	time_cost
time_basis	time_unit	

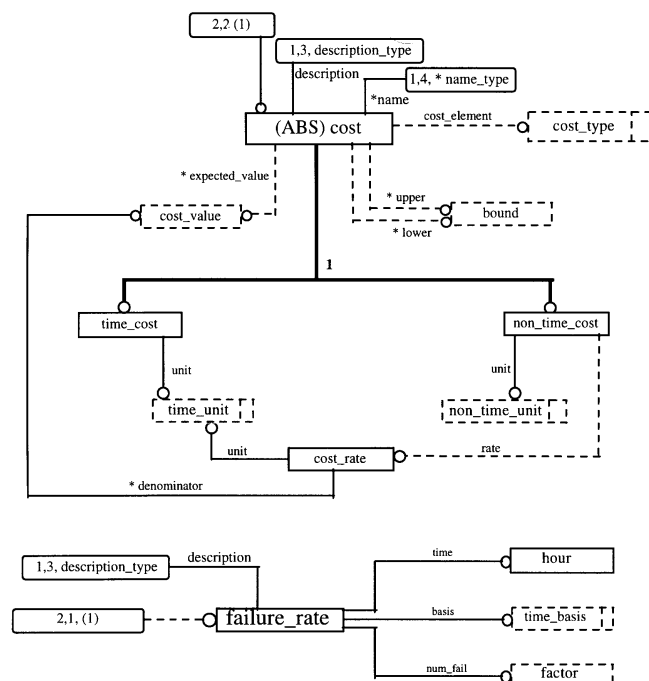
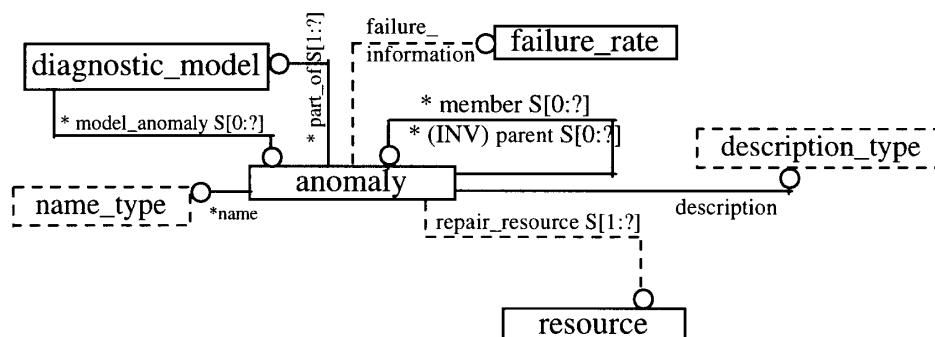


Figure 4-B—Common Element Model (diagram 2 of 2)

5.2.1 anomaly

EXPRESS-G view:



Any diagnosis shall point to some part of the system to be adjusted, calibrated, repaired, replaced, etc. In short, a diagnosis shall lead to the object of a maintenance action. The “physical” entity in the test subject corresponding to the diagnosis (thus to the entity diagnosis) is the entity defined by anomaly, and identifies a failure mode of the test subject. The lattice constructed of anomaly entities shall not be cyclic in the sense that no anomaly be an ancestor of itself.

EXPRESS Specification:

```
TYPE name_type = STRING;
END_TYPE;
```

```
TYPE description_type = STRING;
END_TYPE;
```

```
ENTITY anomaly;
    name                :    name_type;
    description          :    description_type;
    failure_information   :    OPTIONAL failure_rate;
    repair_resource       :    OPTIONAL SET [1:?] OF resource;
    member               :    SET [0:?] OF anomaly;
    part_of              :    SET [1:?] OF diagnostic_model;
```

```
INVERSE
    parent :                SET [0:?] OF anomaly FOR member;
```

UNIQUE

```
    name;
```

WHERE

```
    graph_is_acyclic:
        anomaly_dag(SELF.member,SELF.parent);
    anomaly_listed:
        anomaly_in_list(SELF,SELF.part_of);
```

END_ENTITY;

```
FUNCTION anomaly_dag(member:anomaly;parents:SET [0:?] OF anomaly):
```

```
    LOGICAL;
```

```
    LOCAL
```

```
        grand_parents : SET [0:?] OF anomaly := [];
```

```
    END_LOCAL;
```

```
    IF SIZEOF(parents) = 0 THEN
```

```
        RETURN (TRUE);
```

```
    END_IF;
```

```
    IF member IN parents THEN
```

```
        RETURN (FALSE);
```

```

ELSE
    REPEAT i := LOINDEX(parents) TO HIINDEX(parents);
        grand_parents := grand_parents + parents[i].parent;
    END_REPEAT;
    RETURN(anomaly_dag(member, grand_parents));
END_IF;
END_FUNCTION;

FUNCTION anomaly_in_list(anom:anomaly;
    mdl:SET [1:?] OF diagnostic_model):BOOLEAN;
LOCAL
    check : BOOLEAN;
END_LOCAL;

REPEAT i := LOINDEX(mdl) TO HIINDEX(mdl);
    check := FALSE;
    REPEAT j := LOINDEX(mdl[i].model_anomaly) TO
        HIINDEX(mdl[i].model_anomaly);
        IF (mdl[i].model_anomaly[j] = anom) THEN
            check := TRUE;
        END_IF;
    END_REPEAT;
    IF (check = FALSE) THEN
        RETURN(FALSE);
    END_IF;
END_REPEAT;
RETURN(TRUE);
END_FUNCTION;

```

Attribute definitions are as follow:

name: A unique attribute used to identify the entity.

description: Attribute used to provide an elaborated explanation of what the entity is.

failure_information: An optional value indicating a measure of the expected failure rate of the anomaly. This value can be converted to a probability of failure.

repair_resource: An optional set of resource required to perform the repair associated with this construct. Resources specified by this construct apply to all anomalies that are members of this construct.

member: The set of constituent anomaly of which this entity is composed. This attribute is required but the set can be empty (e.g., if the anomaly is at the lowest level of indenture).

Example: An anomaly might be a failure mode of test subject #4 (Flight Computer), consisting of a failure mode of BOARD #1 (A/D converter), BOARD #2 (CPU), BOARD #3 (Display), etc. BOARD #3 may consist of failure modes of CHIP #1, CHIP #2, CHIP #3 and all may be at different repair levels.

part_of: A pointer to a set of models that reference this anomaly.

parent: The set of anomaly that have the current anomaly as a member. This attribute is required but the set can be empty (e.g., if the anomaly is at the highest level of indenture).

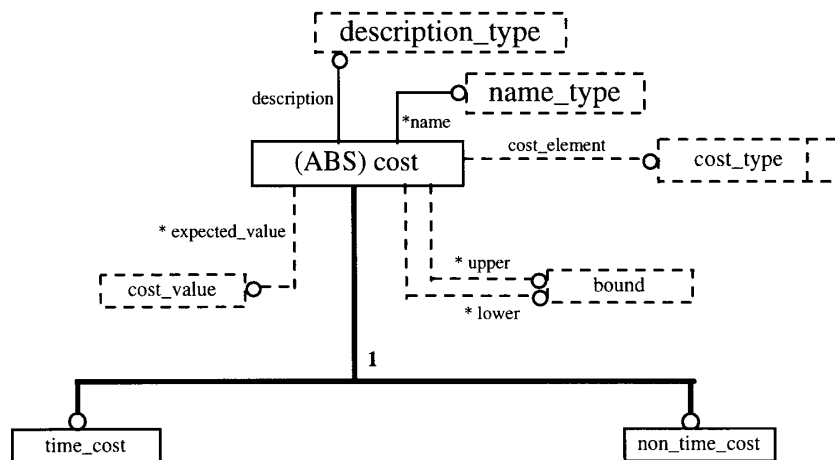
Formal propositions are as follow:

graph_is_acyclic: The structure of anomaly is referred to as a lattice and corresponds to a directed acyclic graph. In other words, traversing the **member** relationships from an anomaly should not result in returning to the same anomaly. This constraint is verified by using the function **anomaly_dag**, which traverses the anomaly structure.

anomaly_listed: Determines whether the current anomaly has been listed in the **diagnostic_models** referenced by the **part_of** attribute. This rule serves as a companion to the **anomaly_is_rollup** rule for **diagnostic_model**.

5.2.2 cost

EXPRESS-G View:



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 the time in seconds 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, reentry.

Each **cost** has an associated unit to enable consistent processing of included values. Two types of units are defined as derived types and include **time_unit** and **non_time_unit**. **Time_unit** is an enumerated type that defines the types of time units to be associated with a time-related cost. **Non_time_unit** is an enumerated type covering predominant international currencies and defines the types of cost units to be associated with non-time-related costs.

EXPRESS Specifications:

TYPE time_unit =

ENUMERATION OF (user_defined, hours, minutes, seconds, msec,
usec, nsec, psec);

END_TYPE;

TYPE non_time_unit = ENUMERATION OF

(user_defined, u_s_dollars, pounds_sterling, deutsche_marks,
yen, french_francs, count);

END_TYPE;

TYPE cost_value = REAL;

END_TYPE;

TYPE bound = REAL;

END_TYPE;

TYPE cost_type =

ENUMERATION OF (user_defined, performance, setup,
access, reentry);

END_TYPE;

ENTITY cost

ABSTRACT SUPERTYPE OF(ONEOF(time_cost, non_time_cost));

name	:	name_type;
description	:	description_type;
expected_value	:	cost_value;
cost_element	:	cost_type;
upper	:	OPTIONAL bound;
lower	:	OPTIONAL bound;

UNIQUE

name;

WHERE

valid_bounds :

(NOT(EXISTS(SELF.upper)) AND NOT(EXISTS(SELF.lower))) XOR
(SELF.lower <= SELF.expected_value <= SELF.upper);

END_ENTITY;

Attribute definitions are as follow:

name: A unique attribute used to identify the entity.

description: An attribute used to provide an elaborated explanation of what the entity is.

expected_value: An attribute providing the nominal, or expected value for the cost entity.

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

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

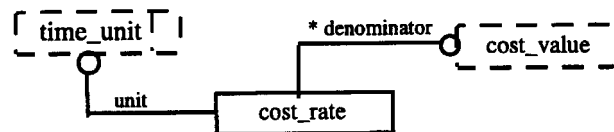
lower: An optional attribute providing the nominal lower bound for the value of the cost entity.

Formal proposition is as follows:

valid_bounds: 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 value of the *upper* attribute. Further, the *expected_value* shall lie between these two bounds.

5.2.3 cost_rate

EXPRESS-G View :



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, for example, to specify labor rates or rental rates.

EXPRESS Specification:

ENTITY *cost_rate*;

denominator : cost_value;

unit : time_unit;

WHERE

denominator_is_non_zero:

SELF.denominator <> 0;

END_ENTITY;

Attribute definitions are as follow:

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

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.”

Example: If we wish to represent cost per day or cost per 24 hours, this can be done as follows:

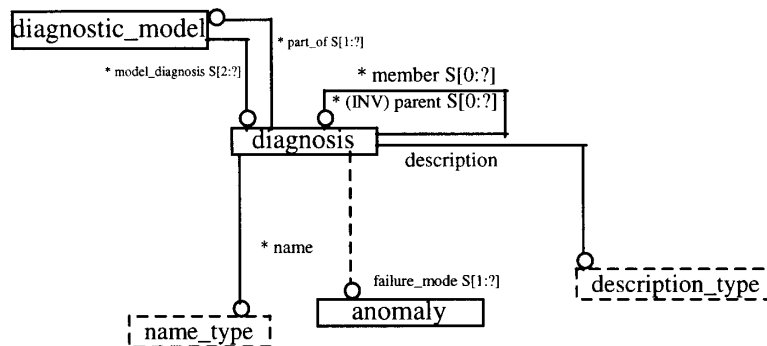
```
pd = cost_rate{denominator -> cost_value{24.0}; unit -> {!hours};};
```

Formal proposition is as follows:

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 non-zero, thus preventing dividing by zero.

5.2.4 diagnosis

EXPRESS-G View:



Diagnosis may be used for the representation of a lattice of diagnostic conclusions. The diagnoses can be interconnected in a lattice to classify constituent units, for example 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.

EXPRESS Specification:

ENTITY **diagnosis**;

```

    name          :    name_type;
    description    :    Description_type;
    failure_mode   :    OPTIONAL SET [1:?] OF anomaly;
    member         :    SET [0:?] OF diagnosis;
    part_of        :    SET [1:?] OF diagnostic_model;
  
```

INVERSE

```

    parent        :    SET [0:?] OF diagnosis FOR member;
  
```

UNIQUE

```

    name;
  
```

WHERE

```

    graph_is_acyclic:
        diagnosis_dag(SELF.member, SELF.parent);
    diagnosis_listed:
        diagnosis_in_list(SELF, SELF.part_of);
  
```

END_ENTITY;

```

FUNCTION diagnosis_dag(member:diagnosis;
    parents : SET [0:?] OF diagnosis):LOGICAL;
LOCAL
    grand_parents : SET [0:?] OF diagnosis := [];
END_LOCAL;
IF SIZEOF(parents) = 0 THEN
    RETURN (TRUE);
END_IF;
IF member IN parents THEN
    RETURN (FALSE);
ELSE
    REPEAT i := LOINDEX(parents) TO HIINDEX(parents);
        grand_parents := grand_parents + parents[i].parent;
    END_REPEAT;
    RETURN(diagnosis_dag(member,grand_parents));
END_IF;
END_FUNCTION;

FUNCTION diagnosis_in_list(diag:diagnosis;
    mdl:SET [1:?] OF diagnostic_model):BOOLEAN;
LOCAL
    check : BOOLEAN;
END_LOCAL;
REPEAT i := LOINDEX(mdl) TO HIINDEX(mdl);
    check := FALSE;
    REPEAT j := LOINDEX(mdl[i].model_diagnosis) TO
        HIINDEX(mdl[i].model_diagnosis);
        IF (mdl[i].model_diagnosis[j] = diag) THEN
            check := TRUE;
        END_IF;
    END_REPEAT;
    IF (check = FALSE) THEN
        RETURN(FALSE);
    END_IF;
END_REPEAT;
RETURN(TRUE);
END_FUNCTION;

```

Attribute definitions are as follow:

name: A unique attribute used to identify the entity.

description: Attribute used to provide an elaborated explanation of what the entity is.

failure_mode: Tie to the physical system as characterized by an anomaly. This correspondence may consist of zero or more anomaly in the system.

member: The set of constituent **diagnosis** of which this entity is composed. This attribute is required but the set can be empty (e.g., if the **diagnosis** is at the lowest level of indenture).

Example: A diagnosis might be of test subject #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.

part_of: A pointer to a set of models that reference this **diagnosis**. This attribute provides a mechanism for linking models based on conclusions drawn during the diagnostic process.

parent: The set of **diagnosis** that have the current **diagnosis** as a member. This attribute is required but the set can be empty (e.g., if the **diagnosis** is at the highest level of indenture).

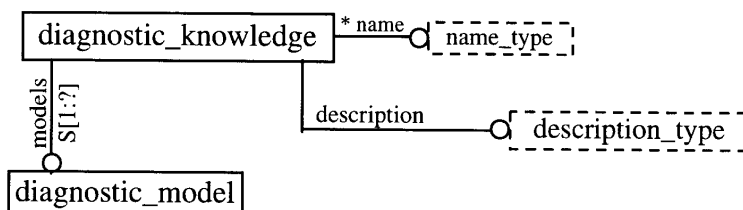
Formal propositions are as follow:

graph_is_acyclic: The structure of **diagnosis** is referred to as a lattice and corresponds to a directed acyclic graph. In other words, traversing the member relationships from a **diagnosis** should not result in returning to the same **diagnosis**. This constraint is verified by using the function **diagnosis_dag**, which traverses the **diagnosis** structure.

diagnosis_listed: A rule that determines whether the current **diagnosis** has been listed in the **diagnostic_models** referenced by the **part_of** attribute. This rule serves as a companion to the **diagnosis_is_rollup** rule for **diagnostic_model**.

5.2.5 diagnostic_knowledge

EXPRESS-G View:



A diagnostic procedure may utilize a number of different AI-ESTATE conformant models. Diagnostic knowledge is the set of AI-ESTATE conformant models that are available to a diagnostic reasoner. By allowing multiple diagnostic models to be specified, an AI-ESTATE implementation can provide the links between these different models.

EXPRESS Specification:

ENTITY `diagnostic_knowledge`;

```

    name          :    name_type;
    description    :    description_type;
    models         :    SET [1:?] OF diagnostic_model;
  
```

UNIQUE

name;

END_ENTITY;

Attribute definitions are as follow:

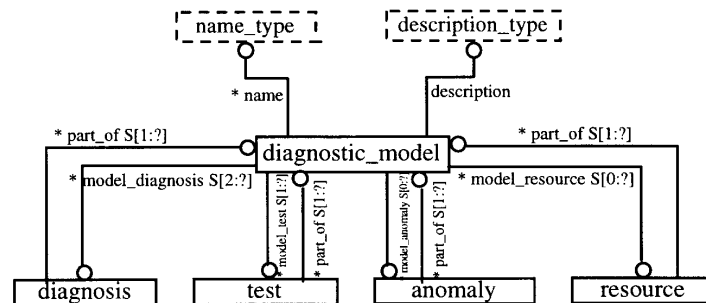
name: A unique attribute used to identify the entity.

description: Attribute used to provide an elaborated explanation of what the entity is.

model: The set of diagnostic models appropriate for the current diagnostic knowledge.

5.2.6 diagnostic_model

EXPRESS-G View:



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 test subject. A `diagnostic_model` refers to some AI-ESTATE conformant model as specified in the annexes of this standard. Entities from the `Common_Element_Model` are referenced as required by those diagnostic models.

EXPRESS Specification:

CONSTANT

no_fault: name_type := 'No Fault';

END_CONSTANT;

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;

UNIQUE

name;

WHERE

one_diagnosis_is_no_fault:
no_fault_in_model(SELF.model_diagnosis);
test_is_rollup:
test_rollup(SELF, SELF.model_test);
diagnosis_is_rollup:
diagnosis_rollup(SELF, SELF.model_diagnosis);
resource_is_rollup:
resource_rollup(SELF, SELF.model_resource);
anomaly_is_rollup:
anomaly_rollup(SELF, SELF.model_anomaly);

END_ENTITY;

FUNCTION no_fault_in_model(diagnoses : SET [2:?] OF diagnosis) :

LOGICAL;

LOCAL

count : INTEGER := 0;
result : LOGICAL := FALSE;

END_LOCAL;

REPEAT i := LOINDEX(diagnoses) TO HIINDEX(diagnoses);

IF diagnoses[i].name = no_fault THEN

count := count + 1;

END_IF;

```

    END_REPEAT;
    IF count = 1 THEN
        result := TRUE;
    END_IF;
    RETURN(result);
END_FUNCTION;

FUNCTION test_rollup mdl:diagnostic_model; tst:SET [1:?] OF test):
    BOOLEAN;
    LOCAL
        check : BOOLEAN;
    END_LOCAL;
    REPEAT i := LOINDEX(tst) TO HIINDEX(tst);
        check := FALSE;
        REPEAT j:= LOINDEX(tst[i].part_of) TO
            HIINDEX(tst[i].part_of);
            IF (tst[i].part_of[j] = mdl) THEN
                check := TRUE;
            END_IF;
        END_REPEAT;
        IF (check = FALSE) THEN
            RETURN(FALSE);
        END_IF;
    END_REPEAT;
    RETURN(TRUE);
END_FUNCTION;

FUNCTION diagnosis_rollup(mdl:diagnostic_model;
    diag:SET [2:?] OF diagnosis):BOOLEAN;
    LOCAL
        check : BOOLEAN;
    END_LOCAL;
    REPEAT i := LOINDEX(diag) TO HIINDEX(diag);
        check := FALSE;
        REPEAT j:= LOINDEX(diag[i].part_of) TO
            HIINDEX(diag[i].part_of);
            IF (diag[i].part_of[j] = mdl) THEN
                check := TRUE;
            END_IF;
        END_REPEAT;
    END_REPEAT;
    RETURN(check);
END_FUNCTION;

```

```

        END_IF;
    END_REPEAT;
    IF (check = FALSE) THEN
        RETURN(FALSE);
    END_IF;
END_REPEAT;
RETURN(TRUE);
END_FUNCTION;

FUNCTION resource_rollup mdl:diagnostic_model;
    res:SET [0:?] OF resource):BOOLEAN;
LOCAL
    check : BOOLEAN;
END_LOCAL;
REPEAT i := LOINDEX(res) TO HIINDEX(res);
    check := FALSE;
    REPEAT j:= LOINDEX(res[i].part_of) TO
        HIINDEX(res[i].part_of);
        IF (res[i].part_of[j] = mdl) THEN
            check := TRUE;
        END_IF;
    END_REPEAT;
    IF (check = FALSE) THEN
        RETURN(FALSE);
    END_IF;
END_REPEAT;
RETURN(TRUE);
END_FUNCTION;

FUNCTION anomaly_rollup mdl:diagnostic_model;
    anom:SET [0:?] OF anomaly):BOOLEAN;
LOCAL
    check : BOOLEAN;
END_LOCAL;
REPEAT i := LOINDEX(anom) TO HIINDEX(anom);
    check := FALSE;
    REPEAT j:= LOINDEX(anom[i].part_of) TO
        HIINDEX(anom[i].part_of);

```

```

        IF (anom[i].part_of[j] = mdl) THEN
            check := TRUE;
        END_IF;
    END_REPEAT;
    IF (check = FALSE) THEN
        RETURN(FALSE);
    END_IF;
END_REPEAT;
RETURN(TRUE);
END_FUNCTION;

```

Attribute definitions are as follow:

name: A unique attribute used to identify the entity.

description: An attribute used to provide an elaborated explanation of what the entity is.

model_test: An attribute used to identify tests that are part of the model.

model_diagnosis: An attribute used to identify diagnoses that are part of the model.

model_resource: An attribute used to identify resources that are part of the model.

model_anomaly: An attributed used to identify anomalies that are part of the model.

Formal propositions are as follow:

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.

test_is_rollup: This rule verifies that all of the **tests** referenced at this level are defined as being part of this model. There is a companion rule for the **tests** to ensure that they are listed in the rollup at this level.

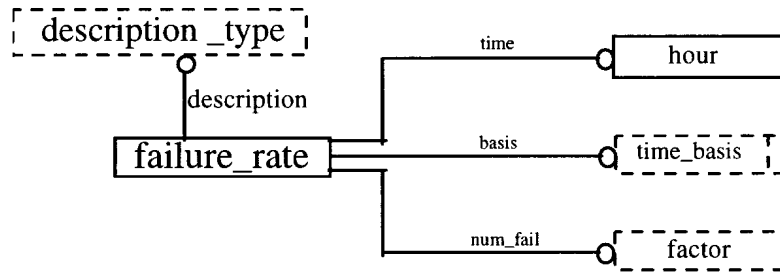
diagnosis_is_rollup: This rule verifies that all of the **diagnosis** referenced at this level are defined as being part of this model. There is a companion rule for the **diagnosis** to ensure that they are listed in the rollup at this level.

resource_is_rollup: This rule verifies that all of the **resources** referenced at this level are defined as being part of this model. There is a companion rule for the **resources** to ensure that they are listed in the rollup at this level.

anomaly_is_rollup: This rule verifies that all of the **anomaly** referenced at this level are defined as being part of this model. There is a companion rule for the **anomaly** to ensure that they are listed in the rollup at this level.

5.2.7 failure_rate

EXPRESS-G View:



This entity indicates the failure rate of an *anomaly*, where the time units are failures per million hours. 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:

CONSTANT

```
per_million:hours := 1000000;
```

END_CONSTANT;

TYPE time_basis = ENUMERATION OF

```
(not_known, operating_time, clock_time);
```

END_TYPE;

TYPE hour = INTEGER;

END_TYPE;

TYPE factor = REAL;

END_TYPE;

ENTITY failure_rate;

```
description      :      description_type;
```

```
basis            :      time_basis;
```

```
time             :      hour;
```

```
num_fail         :      factor;
```

WHERE

```
constant_denominator:
```

```
time = per_million;
```

END_ENTITY;

Attribute definitions are as follow:

description: An attribute that describes the failure_rate entity.

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.

time: An attribute to define the denominator of the failure rate calculation as ‘failures per million hours’.

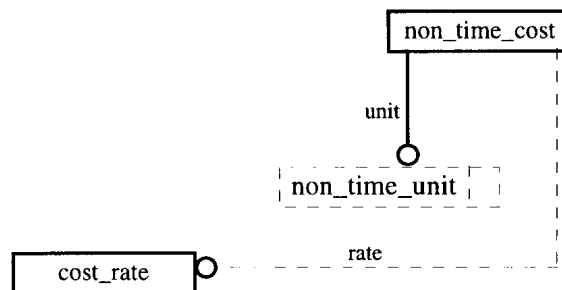
num_fail: An attribute that gives the number of failures.

Formal proposition is as follows:

constant_denominator: A rule that constrains the failure rate to be “per million hours.”

5.2.8 non_time_cost

EXPRESS-G View:



The **non_time_cost** construct represents cost attributes that are not time related and are computed. Financial costs as well as costs computed from an objective function fall into this category. Objective functions typically combine functions of weighted cost parameters that may be temporal, financial, or a quantification of some other expense of the test. Units for objective function-based costs would typically be defined as **user_defined**. The **non_time_cost** entity inherits all attributes from **cost**.

EXPRESS Specification:

```
ENTITY non_time_cost
    SUBTYPE OF (cost);
    unit : non_time_unit;
    rate : OPTIONAL cost_rate;
END_ENTITY;
```

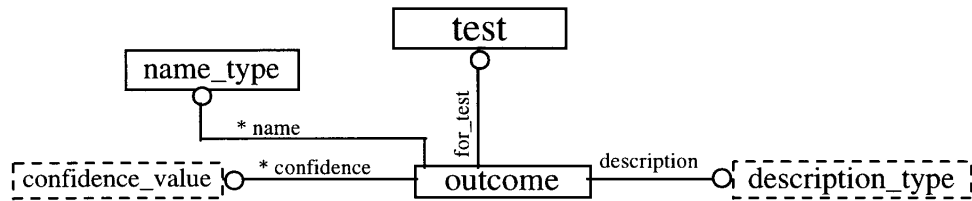
Attribute definitions are as follow:

unit: An attribute to define the non-time units associated with the non-time-related cost.

rate: An optional attribute to provide a specific non time cost to be associated with the rate. In other words, this attribute will provide the cost part of “cost-per-unit-time.”

5.2.9 outcome

EXPRESS-G View:



Associated with any test in a model using the test entity definition is a set of outcomes. Outcomes associate discrete values to ranges of measurements in the test definition and form the basis for the diagnostic process. For any given test in an AI-ESTATE model, at least two **outcomes** shall be defined. Thus, at a minimum, outcomes corresponding, for example, to **pass** and **fail** should be provided with a given test specification. At times more than two outcomes may be appropriate, for example **pass**, **fail-low**, and **fail-high**.

EXPRESS Specification:

```

TYPE confidence_value = REAL;
END_TYPE;

```

```

ENTITY outcome;

```

```

    name      :      name_type;
    description :      description_type;
    confidence :      confidence_value;

```

```

INVERSE

```

```

    for_test :test FOR test_outcome;

```

```

UNIQUE

```

```

    name;

```

```

WHERE

```

```

    range :      (0.0 <= SELF.confidence <= 1.0);

```

```

END_ENTITY;

```

Attribute definitions are as follow:

name: A unique attribute used to identify the entity.

description: An attribute used to provide an elaborated explanation of what the entity is.

confidence: A measure of the confidence in the test outcome based on characteristics of performing the test. This measure is used for providing confidence measures in the diagnoses.

for_test: Links the outcome to the specific test with that outcome.