IEC 62243

Edition 2.0 2012-06

INTERNATIONAL STANDARD

IEEE Std 1232™

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

IEC 62243:2012(E) IEEE Std 1232-2010

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

THIS PUBLICATION IS COPYRIGHT PROTECTED
Copyright © 2010 IEEE

All rights reserved. IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Inc.

Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the IEC Central Office.

Any questions about IEEE copyright should be addressed to the IEEE. Enquiries about obtaining additional rights to this publication and other information requests should be addressed to the IEC or your local IEC member National Committee.

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue
New York, NY 10016-5997
United States of America
stds.info@ieee.org
www.ieee.org

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,…).
It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

IEC 62243

Edition 2.0 2012-06

INTERNATIONAL STANDARD

IEEE Std 1232™

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

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

PRICE CODE XG

ICS 25.040; 35.060

ISBN 978-2-83220-102-2

Warning! Make sure that you obtained this publication from an authorized distributor.

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

Contents

1. Overview 1
1.1 Scope 2
1.2 Purpose 2
1.3 Conventions used in this document 3
1.4 IEEE download site 3

2. Normative references 3

3. Definitions, acronyms, and abbreviations 4
3.1 Definitions 4
3.2 Acronyms and abbreviations 5

4. Description of AI-ESTATE 5
4.1 AI-ESTATE architecture 5
4.2 Binding strategy 8

5. AI-ESTATE usage 9
5.1 Interchange format 9
5.2 Extensibility 11
5.3 Conformance 11

6. Models 12
6.1 AI_ESTATE_CEM 12
6.2 AI_ESTATE_BNM 55
6.3 AI_ESTATE_DIM 64
6.4 AI_ESTATE_DLM 68
6.5 AI_ESTATE_FTM 72
6.6 AI_ESTATE_DCM 77

7. Reasoner manipulation services 92
7.1 Service order dependence 92
7.2 Status codes 95
7.3 Data types for the reasoner manipulation services 95
7.4 Required services 110
7.5 Optional services 132

Annex A (informative) Bibliography 136

Annex B (informative) Overview of EXPRESS 138

Annex C (informative) Overview of ISO 10303-28:2007 145

Annex D (informative) Overview of ISO 10303-21:1994 152

Annex E (normative) Information object registration 157

Annex F (normative) Universal resource names for derived XML schemas 158

Annex G (informative) IEEE List of Participants 159

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

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

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation.

   IEEE Standards documents are developed within IEEE Societies and Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. IEEE develops its standards through a consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of IEEE and serve without compensation. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards. Use of IEEE Standards documents is wholly voluntary. IEEE documents are made available for use subject to important notices and legal disclaimers (see http://standards.ieee.org/IPR/disclaimers.html for more information).

   IEC collaborates closely with IEEE in accordance with conditions determined by agreement between the two organizations.

2) The formal decisions of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees. The formal decisions of IEEE on technical matters, once consensus within IEEE Societies and Standards Coordinating Committees has been reached, is determined by a balanced ballot of materially interested parties who indicate interest in reviewing the proposed standard. Final approval of the IEEE standards document is given by the IEEE Standards Association (IEEE-SA) Standards Board.

3) IEC/IEEE Publications have the form of recommendations for international use and are accepted by IEC National Committees/IEEE Societies in that sense. While all reasonable efforts are made to ensure that the technical content of IEC/IEEE Publications is accurate, IEC or IEEE cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications (including IEC/IEEE Publications) transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC/IEEE Publication and the corresponding national or regional publication shall be clearly indicated in the latter.

5) IEC and IEEE do not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC and IEEE are not responsible for any services carried out by independent certification bodies.

6) All users should ensure that they have the latest edition of this publication.

7) No liability shall attach to IEC or IEEE or their directors, employees, servants or agents including individual experts and members of technical committees and IEC National Committees, or volunteers of IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board, for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC/IEEE Publication or any other IEC or IEEE Publications.

8) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

9) Attention is drawn to the possibility that implementation of this IEC/IEEE Publication may require use of material 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. IEC or IEEE shall not be held responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patent Claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

International Standard IEC 62243/ IEEE Std 1232-2010 has been processed through IEC technical committee 93: Design automation, under the IEC/IEEE Dual Logo Agreement.

This second edition cancels and replaces the first edition, published in 2005, and constitutes a technical revision.

The text of this standard is based on the following documents:

IEEE Std	FDIS	Report on voting
IEEE Std 1232-2010	93/320/FDIS	93/327/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

The IEC Technical Committee and IEEE Technical Committee have decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

IEEE Std 1232™-2010
(Revision of
IEEE Std 1232-2002)

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

Sponsor

IEEE Standards Coordinating Committee 20 on Test and Diagnosis for Electronic Systems

Approved 8 December 2010

IEEE-SA Standards Board

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

Abstract: Data interchange and standard software services for test and diagnostic environments are defined by Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE). The purpose of AI-ESTATE is to standardize interfaces for 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. The services to control a diagnostic reasoned are defined by this standard.

Keywords: AI-ESTATE, Bayesian Network, diagnosis, diagnostic inference, diagnostic model, diagnostic services, D-matrix, fault tree, IEEE 1232, knowledge exchange, system test

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Incorporated.

W3C is a registered trademark of the W3C® (registered in numerous countries) World Wide Web Consortium. Marks of W3C are registered and held by its host institutions MIT, ERCIM, and Keio.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

IEEE Introduction

This introduction is not part of IEEE Std 1232-2010, IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE).

The AI-ESTATE standard provides a formal framework for exchanging diagnostic knowledge and communicating with 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 applications can be developed to communicate with diagnostic reasoners in a consistent and reliable way.

Notice to users

Laws and regulations

Users of these documents should consult all applicable laws and regulations. Compliance with the provisions of this standard does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

This document is copyrighted by the IEEE. It is made available for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

Updating of IEEE documents

Users of IEEE standards should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Standards Association web site at http://ieeexplore.ieee.org/xpl/standards.jsp, or contact the IEEE at the address listed previously.

For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA web site at http://standards.ieee.org.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/reading/ieee/updates/errata/index.html. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

Patents

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 is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

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

IMPORTANT NOTICE: This standard is not intended to ensure safety, security, health, or environmental protection. Implementers of the standard are responsible for determining appropriate safety, security, environmental, and health practices or regulatory requirements.

This IEEE document is made available for use subject to important notices and legal disclaimers. These notices and disclaimers appear in all publications containing this document and may be found under the heading "Important Notice" or "Important Notices and Disclaimers Concerning IEEE Documents." They can also be obtained on request from IEEE or viewed at http://standards.ieee.org/IPR/disclaimers.html.

1. Overview

The Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE) standard was developed by the Diagnostic and Maintenance Control Subcommittee of the IEEE Standards Coordinating Committee 20 (SCC20) on Test and Diagnosis for Electronic Systems to serve as a standard for defining interfaces among diagnostic reasoners and 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.

One approach to defining the interfaces for a component of a larger system is to model, formally, the information being passed across the system's interfaces. Such a model is known as an "information model." The purpose of an information model is to identify clearly the objects in a domain of discourse (e.g., diagnostics) to enable precise communication about that domain. Such a model comprises objects or entities, relationships between those objects, and constraints on the objects and their relationships. When taken together, elements provide a complete, unambiguous, formal representation of the domain of discourse. In other words, they provide a formal language for communicating about the domain.

Using information models, information exchange can be facilitated in two ways. The first is through a set of exchange files. Specifically, information can be stored by one application in a file and read by a second application. The file format is derived directly from the information model and defines the syntax of the message contained within it. The semantics of the message (i.e., the legal content of the file) is defined by the semantics of the model. The second means of information exchange is through a set of services defined for a system component as accessed via the communications backbone. The interface definition for the component is derived from the information model and defines the syntax of the message. Once again, the legal content of the message is defined by the semantics of the model.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

The semantics of information models are provided in two ways. First, the model itself defines a machine-readable semantic structure and associated constraints that ensure consistent exchange and processing of the concepts and relationships of the model elements. Second, human-readable definitions specify the correct interpretation of the model elements.

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

— Enable the consistent exchange and integration of diagnostic capabilities

AI-ESTATE defines key data and knowledge specification formats. No host computer dependence is contained in the AI-ESTATE standard. Systems that use only these specification formats will be portable. This does not preclude use of AI-ESTATE interfaces with nonconformant specification formats; however, such systems may not be portable. A diagnostic model can be moved from one AI-ESTATE implementation to another by translating it into one of two interchange formats described in the specification. 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 or its own internal form.

Software specifications defined in this standard provide a consistent means of communicating with diagnostic reasoners through a well-defined set of services. This supports interoperability of diagnostic reasoner with other elements of a test environment with no effect on the other elements of the system.

This standard also provides an extension mechanism to allow the inclusion of new diagnostic technology outside the scope of the Al-ESTATE specification.

An overview of EXPRESS can be found in Annex B. Overviews of the ISO 10303-28:2007[1] and ISO 10303-21:1994 exchange formats can be found in Annex C and Annex D, respectively.

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 testing and diagnosis. The standard defines formal information models and software services specific to several different types of diagnostic reasoners.

[1]Information on references can be found in Clause 2.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

The purpose is to provide semantically sound definitions of diagnostic knowledge and to specify software exchange and service interfaces that are consistent with the state of the practice in modern test and diagnostic systems (e.g., the use of eXtensible Markup Language [XML] and web services).

1.3 Conventions used in this document

This standard specifies information models, exchange formats, and services using the EXPRESS language and uses the following conventions in their presentation.

Information models are provided in the form of EXPRESS schemas. Exchange files provide the instances of those schemas for a particular diagnostic model. Note that "information model" and "diagnostic model" use the word "model" in subtly different ways. In an attempt to resolve this confusion, in this document, information models will be referred to as EXPRESS schemas and instances of a schema corresponding to a diagnostic model will be referred to as instances (e.g., Dynamic Context Part 21 instance).

All specifications in the EXPRESS and XML languages are given in the `Courier New` type font. The EXPRESS schemas include comment delimiters "(*" and "*)".

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 the event of conflict between this standard and a related standard such as IEEE Std 1636™-2009 [B5],[2] the standard as it applies to the information being produced shall take precedence. In the event of any conflict between the models and AI-ESTATE definitions (Clause 3), the models' lexical definitions shall take precedence

1.4 IEEE download site

The schemas and examples that accompany this standard are available on the Internet at http://standards.ieee.org/downloads/1232/1232-2010.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

Internet Engineering Task Force (IETF) RFC 2396 (August 1998), Uniform Resource Identifiers (URI): Generic Syntax. [cited 2004-03-15].[3,4]

[2] The numbers in brackets correspond to those of the bibliography in Annex A.
[3] IETF publications are available from the Internet Engineering Task Force, IETF Secretariat, c/o Association Management Solutions, LLC (AMS), 48377 Fremont Boulevard, Suite 117, Fremont, CA 94538, USA (http://www.ietf.org).
[4] This reference can be downloaded at http://www.ietf.org/rfc/rfc2396.txt.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

ISO 10303-11:1994 Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 11: The EXPRESS Language Reference Manual.[5]

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

ISO 10303-21:1994 Technical Corrigendum 1.

ISO 10303-28:2007 Industrial Automation Systems and Integraion—Product Data Representation and Exchange—Part 28: XML Representation of EXPRESS Schemas and Data using XML Schemas.

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. *The IEEE Standards Dictionary: Glossary of Terms & Definitions* should be consulted for terms not defined in this clause.[6]

ambiguity group: A set of diagnoses that cannot be distinguished with the given set of test outcomes.

diagnostic reasoner: A system that uses a knowledge base to infer conclusions.

diagnostic strategy: (**A**) An approach taken to combine factors including constraints, goals and other considerations to be applied to the localization of faults in a system. (**B**) The approach taken to evaluate a system in order to obtain a diagnostic result.

EXPRESS schema: A specification of data types, structural constraints, and algorithmic rules corresponding to some domain of interest.

eXtensible Markup Language (XML) schema: A specification of a type of XML document typically expressed in terms of constraints of structure and content of documents of that type, above and beyond the basic syntactical constraints imposed by XML itself.

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

information model: A specification of a set of objects in a domain of discourse to enable precise and unambiguous communication about that domain. Such a model consists of one or more schemata, each of which comprise objects or entities, relationships between those objects, and constraints on the objects and their relationships.

instance: An occurrence of a realized schema or schema element.

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

[5]ISO publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, 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, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).
[6] *The IEEE Standards Dictionary: Glossary of Terms & Definitions* is available at http://shop.ieee.org/.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

knowledge base: A set of data, data semantics and relationships, and functions used by diagnostic reasoners.

native format: Data that exist in a format either produced or consumed by some non–AI-ESTATE diagnostic reasoner.

UOS document: A document that conforms to a single governing EXPRESS schema and follows Part 28's default mapping from EXPRESS to eXtensible Markup Language (XML).

3.2 Acronyms and abbreviations

AI-ESTATE	Artificial Intelligence Exchange and Service Tie to All Test Environments
BNM	Bayesian Network Model
CDF	cumulative distrubution function
CEM	Common Element Model
DAG	directed acyclic graph
DCM	Dynamic Context Model
DIM	Dmatrix Inference Model
DLM	Diagnostic Logic Model
FTM	Fault Tree Model
PDF	probability distribution function
SCC20	Standards Coordinating Committee 20
UOS	unit of serialization
UUT	unit under test
W3C®	World Wide Web Consortium
XML	eXtensible Markup Language

4. Description of AI-ESTATE

4.1 AI-ESTATE architecture

This standard provides the following:

— An overview of the AI-ESTATE architecture

— A formal definition of diagnostic models for systems under test

— Formal definitions of interchange formats for exchange of diagnostic models

— A formal definition of 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 diagnostic reasoners. The approach taken to address this need is by providing interchangeable files. The second aspect concerns the need for an AI-ESTATE conformant diagnostic reasoner to interact and interoperate with other elements in a test environment (see Figure 1).

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

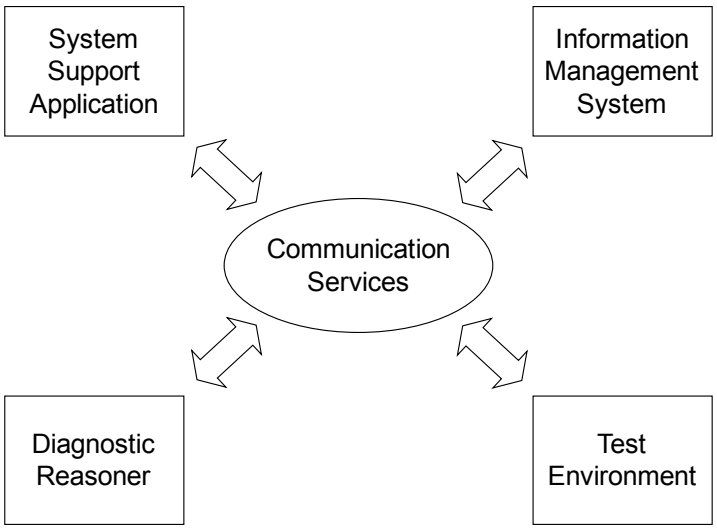

Figure 1—AI-ESTATE architectural concept

Services are provided by AI-ESTATE conformant systems to the other functional elements of a test environment. Reasoners may include (but are not necessarily limited to) diagnostic systems, test sequencers, maintenance data feedback analyzers, intelligent user interfaces, and intelligent test programs. AI-ESTATE specifically focuses on diagnostic reasoners. Thus, services may be provided by a diagnostic reasoner to the system support application, an information management system, and the test environment. The reasoner will use services provided by these other systems as required. Note that services provided by these other systems are not specified by the AI-ESTATE standard.

Data interchange formats are specified to provide a means for exchanging knowledge bases between diagnostic reasoners without the need to apply an information management system. This standard facilitates the use of standard representations of diagnostic data and knowledge within the context of an AI-ESTATE conformant diagnostic reasoner. 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 (CEM) 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), RepairItem (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.

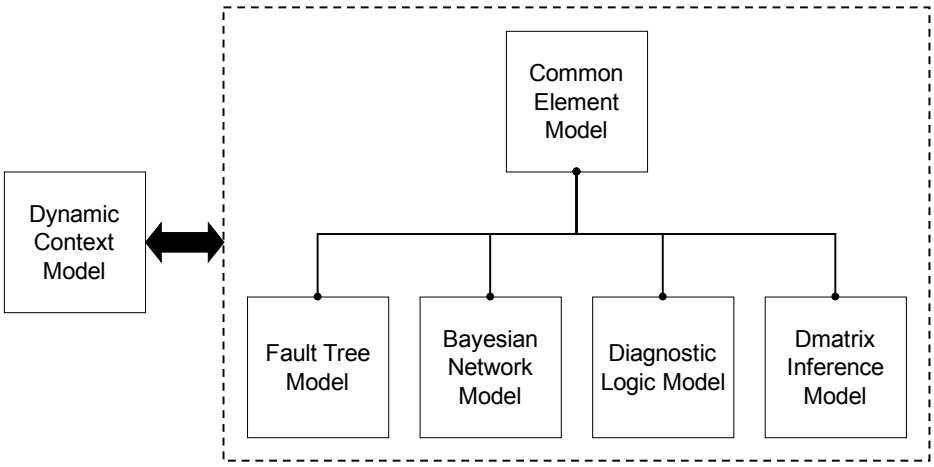

Figure 2—Hierarchical structure of the AI-ESTATE models

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

Below the Common Element Model in Figure 2 is a set of data and knowledge models (i.e., the Bayesian Network Model [BNM], the Diagnostics Logistic Model [DLM], the Dmatrix Inference Model [DIM], and the Fault Tree Model [FTM]) that specialize the constructs in the Common Element Model and tailor the constructs to the application's particular reasoning requirements. 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. As indicated by the dotted line in Figure 2, the Dynamic Context Model (DCM) does not specialize but rather interacts with the CEM. The DCM defines entities that represent the context and history of the reasoning process and defines the interface by which that information can be exchanged.

The models and services for AI-ESTATE utilize four levels of abstraction related to the definition and use of information in a diagnostic application. These four levels are described as follows:

a) A definition is the specification of some entity or concept within the AI-ESTATE domain. A definition encapsulates all of the information that constitutes an entity or concept. For example, AI-ESTATE defines the concept of a "Test" as an entity definition in the Common Element Model.

b) An instance is the static realization of an entity or concept definition. For example, a specific test used by a diagnostic application may be created in a diagnostic model and includes values defining the test (e.g., name, description, and the set of available outcomes).

c) An occurrence is the dynamic realization of an instance against a timeline. The occurrence maintains, within its scope, all of the information necessary to evaluate the instance at the time it is valid. For example, when a sequence of tests has been specified to be performed, it is said that the tests in that sequence "occur" in the scope of the corresponding timeline.

d) An execution is a historical trace of the information that has been collected by occurrences over a period of time. An execution is recorded when the test is actually performed. At that time, specific information related to the performance and results of the test can be captured.

Within the AI-ESTATE architecture, the information models specified in Clause 6 provide the definition of the information. Diagnostic models that conform to the specifications in Clause 6 are the instances of these information models. The instances of the model entities occurring in the application state flow specified in 7.1 correspond to the occurrence of the entities in a diagnostic process. Finally, the record of the occurrence of these entities collected from services performed in the diagnostic process corresponds to the execution (or execution trace) of the session. The structure for exporting this execution trace is defined by the DCM in Clause 6.

As illustrated in Figure 3, this standard also defines the software services to be provided by an AI-ESTATE conformant diagnostic reasoner. All of the services are defined relative to the entities and attributes of the information models and comprise the diagnostic reasoner interface. As can be seen in Figure 2, each of the elements that interface with the reasoner will provide its own set of services to the other system components, but those service definitions are beyond the scope of this document.

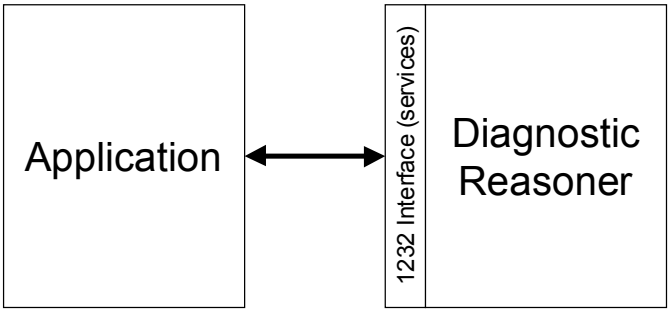

Figure 3—AI-ESTATE interface layer

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

The service definitions encapsulate the reasoner so that the underlying implementation details are hidden from the diagnostic system clients. Such services encompass an abstraction of that behavior that is common to all diagnostic reasoners, regardless of implementation details. Therefore, it is the mechanism of encapsulation that provides for the interchangeability of AI ESTATE conformant diagnostic reasoners within a system.

A reasoner implementation shall provide, at the least, a status indicator (see 7.2) as a response to any service request defined by this specification. The Diagnostic Reasoner shall provide the required services specified by this standard to a client. The diagnostic reasoner shall only utilize a single model during a session.

4.2 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 this standard; 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 shall 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 shall be hidden from the client. This implies that the binding layer provides data type definitions as specified in this standard. It is beyond the scope of this standard to define bindings for each implementation language. However, in the interest of interoperability, the standard provides the following guidance for services passing and returning data:

- Component implementations should use messages in their native format.
- Object-oriented implementations should use objects.
- Procedural implementations should use structures.
- Other implementations should use XML entities defined by Part 28 schemas.

The application and diagnostic reasoner programs may be written in different languages as long as the translation is handled transparently by 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 initializeDiagnosticProcess service as specified in EXPRESS:

```
FUNCTION initializeDiagnosticProcess(
   itemID : Identifier,
   repairItemName : NameType) :  NameType;
END_FUNCTION
```

It has the name initializeDiagnosticProcess, accepts two arguments, one of type Identifier and one of NameType, and returns a NameType. The declaration of a corresponding binding function written in C would be:

```
NameType initializeDiagnosticProcess (Identifier itemID, NameType repairItemName);
```

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

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,

```
NameType initializeDiagnosticProcess (Identifier itemID, NameType repairItemName)
{
  NameType name;
.
.
.
  name = . . .;

.
.
  return name;
}
```

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

```
typedef char*  NameType;
typedef char*  Identifier;
```

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

5. AI-ESTATE usage

5.1 Interchange format

AI-ESTATE models are specified to facilitate data interchange in the context of test and diagnosis. The interchange format permits exchange of diagnostic models using a neutral format, thus providing portability of diagnostic knowledge across applications. The following two interchange formats are specified by AI-ESTATE:

— ISO 10303-21:1994

— ISO 10303-28:2007

5.1.1 ISO 10303-21 Exchange Format

AI-ESTATE exchange files that use the ISO 10303-21 format shall adhere to ISO 10303-21:1994 Technical Corrigendum 1 (known as version “2”) and conformance class “1” (known as the internal mapping). The ISO version of ISO 10303-21:1994 and the conformance class of the exchange file shall be indicated in the exchange file header using the syntax prescribed by ISO 10303-21:1994.

AI-ESTATE exchange files in the ISO 10303-21 format shall meet the “schema conformance” requirements specified in ISO 10303-21:1994. Schema conformance shall be with respect to one of the AI-ESTATE exchange schemas listed in 5.3.2, referred to as the governing schema of the file. In addition, the data in the exchange file shall adhere to the semantic definitions and requirements specified for the governing schema in Clause 6.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

Given this exchange format, an AI-ESTATE exchange file shall conform to exactly one AI-ESTATE schema, the data in the exchange file shall exist in a single DATA section in the file, and the data in the file shall constitute a "valid population" for the schema. That is, one cannot aggregate data that conforms to different schemas into a single exchange file, nor can one split exchange data across multiple files.

The header of the exchange file shall also identify the unique object identifier of the governing AI-ESTATE schema, using the syntax described in ISO 10303-21:1994. Annex E of this standard lists the globally unique object identifiers assigned to exchange schemas in this version of AI-ESTATE. The object identifier unambiguously identifies the governing AI-ESTATE schema including its version.

5.1.2 ISO 10303-28 exchange format

AI-ESTATE exchange files that use the ISO 10303-28 format shall use the version: ISO 10303-28:2007. This version of ISO 10303-28:2007 provides a great deal of flexibility in how data is mapped to exchange files. To preserve harmony with the ISO 10303-21 exchange format, exchange files that use the ISO 10303-28 format shall adhere to the following constraints:

— AI-ESTATE exchange files in the ISO 10303-28 format shall meet the "iso-10303-28 document" conformance class requirements defined in ISO 10303-28:2007.

— Using the syntax prescribed in ISO 10303-28:2007, the exchange file shall specify exactly one AI-ESTATE EXPRESS schema that governs all of the data in the file.

— Using the syntax prescribed in ISO 10303-28:2007, the exchange file shall indicate that the entity of data in the file constitutes a "valid population" for the governing AI-ESTATE schema. The file shall not contain data that is not in the population. No subset of the population shall be external to the file.

— The exchange file shall contain exactly one "substitute unit of serialization (UOS) element" that contains all the data that are being exchanged.

— The data in the substitute UOS element shall adhere to the "default mapping" from EXPRESS to XML defined in iso-10303-28.

The data in the ISO 10303-28 formatted exchange file shall be governed by one of the AI-ESTATE EXPRESS exchange schemas listed in Clause 6, referred to as the governing schema. In addition, the data in the exchange file shall adhere to the semantic definitions and requirements specified for the governing schema in Clause 6.

AI-ESTATE exchange files in the ISO 10303-28 format shall indicate the target namespace for the data in the file using the syntax specified in ISO 10303-28:2007. The target namespace shall correspond to the governing AI-ESTATE schema. Annex F lists the namespaces assigned to the AI-ESTATE schemas. The namespace unambiguously identifies the governing AI-ESTATE schema and version.

In addition to validation with respect to the governing AI-ESTATE EXPRESS schema, ISO 10303-28:2007 requires XML validation with respect to a "derived XML schema," which is an XML schema that is derived from the governing EXPRESS schema according to rules specified in ISO 10303-28:2007. Derived XML schemas for AI-ESTATE are available on the Internet at http://standards.ieee.org/downloads/1232/1232-2010. Alternatively, users may generate a derived XML schema using the default mapping rules in ISO 10303-28:2007. The derived XML schemas necessarily import several XML schemas that are defined within the ISO 10303-28:2007 standard; these are not available from the IEEE download site.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

5.2 Extensibility

Users of AI-ESTATE may utilize native formats that contain information beyond what is specified by this standard and may recast their native model formats to standard AI-ESTATE format for purposes of conformant exchange. Should extensions be exchanged, an EXPRESS schema shall define those extensions. That schema shall extend an existing AI-ESTATE schema, but shall not redefine concepts that have been defined in the standard EXPRESS schemas. Extensions to AI-ESTATE model entities and newly defined entities will not be recognized as conforming to the standard.

Any application can provide services beyond those defined in this standard and should adhere to the status codes defined in 7.2. These services will not be recognized as conforming to the standard.

Any implemented extensions shall be fully documented to include EXPRESS schemas. All documentation and schemas shall be submitted to the recipient of the data and should also be submitted to the IEEE SCC20 DMC subcommittee.

5.3 Conformance

This subclause defines the requirements for conformance with this standard for diagnostic reasoner application services as well as exchange model instances.

5.3.1 Diagnostic reasoner application services

Diagnostic reasoner applications shall be conformant to all required services in Clause 7. Applications shall also document any deviations from conformance for the benefit of interoperability.

A conformant diagnostic reasoner shall consume conformant exchange model instances of at least one of the BNM, DIM, DLM, and FTM in addition to the CEM. A conformant diagnostic reasoner shall also produce and consume conformant exchange model instances of the DCM as specified in Clause 6. The reasoner shall be able to consume all required and optional model elements. The reasoner shall also be able to consume extended model instances that conform to 5.2. The reasoner may ignore any such extensions. A conformant exchange model instance is defined in 5.3.2.

5.3.2 Exchange model instances

A file shall conform as an exchange model instance for this standard if it satisfies all the following conditions:

- The data set encoded in the file conforms as specified in ISO 10303-11:1994 to one of the EXPRESS information models defined within this standard, designated the governing schema for the instance: BNM,DIM, DLM, FTM, or DCM (each of which includes the CEM).
- The data set in the file consists of a valid instantiation of a single governing schema (i.e., the data set will validate against the governing schema).
- Any extensions represented in the file conform to 5.2.
- The data set in the file adheres to the semantic definitions of the governing schema.

The file is encoded in one of the exchange formats specified in 5.1 per the requirements specified Clause 5.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

6. 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.

(* EXPRESS Specification starts here. *)

(*

6.1 AI_ESTATE_CEM

The AI-ESTATE Common Element Model information model permits the definition of the form and relationships of systems under test and tests at their most basic level. The CEM defines data types and relationships that are common to all static reasoner models within AI-ESTATE. The CEM itself is not an exchange model schema. Rather, the other static model information models import and extend the CEM to add the essential logic for generating diagnostic conclusions. The CEM schema is also interfaced into the DCM information model. See 6.6 for a description how the CEM plays a role in the DCM.

NOTE—One may think of the CEM as the supertype of the static model information models.[7]

Principal components include system items (which are subtyped as repair items and function items), actions (which are subtyped as tests and repairs), and diagnoses (which are subtyped as faults and failures). The common element information model also permits the specification of cost and failure rate information. The information specified in the common element information model is intended to provide the fundamental elements for diagnostics models that are defined as additional information models.

The CEM does not act as an exchange mechanism by itself. However, the data types defined in the CEM are used by the other EXPRESS schemas within this standard. The data types defined in the CEM are common to multiple other EXPRESS schemas, thus resulting in their definition here. For example, the CEM defines DiagnosticModel, which is a supertype to all of the technology-specific diagnostic models (e.g., fault tree).

EXPRESS specification:

```
*)
SCHEMA AI_ESTATE_CEM;
(*
```

6.1.1 CONSTANT

EXPRESS specification:

```
*)
  CONSTANT
    NoFault  : STRING := 'No Fault';
  END_CONSTANT;
(*
```

A constant corresponding to a special diagnosis indicating there is no fault. This constant is used with the associated name attribute of the no-fault diagnosis. With this diagnosis, associated outcomes have different semantics from those specified with the outcome values themselves. Specifically, GOOD means the

[7] Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

reasoner has conclusive evidence that there is no fault, BAD means the reasoner has evidence that at least one fault is present, CANDIDATE means the reasoner has evidence that no fault is present but the evidence is not conclusive, NOTKNOWN means the reasoner has negligble evidence or conflicting evidence about whether or not any fault is present, and USERDEFINED is unspecified for this diagnosis.

6.1.2 ConfidenceValue

Type ConfidenceValue defines a type for specifying a numeric representation for the degree of certainty in the validity of some value or relation between 0 and 1, where 0 is absolute uncertainty and 1 is absolute certainty. The actual application of confidence values is implementation specific.

EXPRESS specification:

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

Formal propositions:

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

6.1.3 CostValue

Type CostValue defines a type for representing a numerical expense or penalty.

EXPRESS specification:

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

6.1.4 DescriptionType

Type DescriptionType defines a type used to provide descriptive text for an entity within the model.

EXPRESS specification:

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

6.1.5 DistributionPoint

Type DistributionPoint defines a real-valued data type for representing a specific value at a specific point in time in a probability distribution.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

EXPRESS specification:

```
*)
  TYPE DistributionPoint = REAL;
  WHERE
    nonNegative     :   (SELF >= 0.0);
  END_TYPE;
(*
```

Formal propositions:

nonNegative	Proposition nonNegative ensures that the distribution point is greater than or equal to 0.

6.1.6 FailureRate

Type FailureRate defines a single values corresponding to the static failure rate associated with a particular fault or failure. The failure rate shall be specified “per million hours.”

EXPRESS specification:

```
*)
  TYPE FailureRate = DistributionPoint;
  END_TYPE;
(*
```

6.1.7 NameType

Type NameType defines a type used to provide an identifying name for an entity within the model. The name is an identifier that is also human readable.

EXPRESS specification:

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

6.1.8 ProbabilityValue

Type ProbabiltyValue defines a real-value constrained to be in the range 0 ... 1 that represents the probability of occurrence of some event.

EXPRESS specification:

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

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

Formal propositions:

validRange Proposition validRange ensures the value is restricted to be in the range 0 ... 1.

6.1.9 QualifierType

Type QualifierType defines a string qualifier for providing a finer grained specification of test outcomes. As a string, it permits application-specific labels to be used.

EXPRESS specification:

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

6.1.10 TimeValue

Type TimeValue defines a real-valued data type for indicating the time at which some event occurs relative to some initial or prior time.

EXPRESS specification:

```
*)
   TYPE TimeValue = REAL;
   WHERE
      nonNegative      :   (SELF >= 0.0);
   END_TYPE;
(*
```

Formal propositions:

nonNegative Proposition nonNegative ensures that the time value is not less than zero.

6.1.11 ActionCostType

Enumerated type ActionCostType defines a type for categorizing the expense or penalty associated with the cost of an action in the diagnostic process. Current enumerated values include the following:

USER_DEFINED_COST : an unspecified application-specific cost

PERFORMANCE : the expense to execute the action

SETUP : the expense to prepare for the action

ACCESS : the expense associated with entry to the location where the action is to be performed

REENTRY : the expense to access a location where the action is to be performed given the same action was previously performed within the current session

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

CONSUMABLE : the expense associated with replenishable material

ROTABLE : the expense associated with in-place repairs to equipment

REPAIRABLE : the expense associated with items that are repaired and returned to inventory

EXPRESS specification:

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

6.1.12 NonTimeUnit

Enumerated type NonTimeUnit defines legal units for costs other than time. Non-time costs are typically incurred in terms of monetary amounts, skill level, training, or counts of some expendable; however, the user-defined value allows for other types of non-time costs to be included.

EXPRESS specification:

```
*)
   TYPE NonTimeUnit = ENUMERATION OF
      (USER_DEFINED_NON_TIME,
      COUNT,
      CURRENCY,
      SKILL,
      TRAINING);
   END_TYPE;
(*
```

6.1.13 OutcomeValues

Enumerated type OutcomeValues provides a list of legal outcomes for actions, tests, and diagnoses.

Aborted : Serves as a constant for action outcomes indicating an action was attempted but not completed.

Bad : Serves as a constant for outcomes indicating a bad or negative state. Specifically, the reasoner has test evidence indicating the associated diagnosis is present.

Candidate : Serves as a constant for outcomes indicating a candidate state. Specifically, there is no test evidence indicating the associated diagnosis is not present, and there is test evidence to suspect the associated diagonsis might be present. As a member of a set of candidate diagnosis, the set should be interpreted as a disjoint group where any one or more of the candidates is present. The distinction between Candidate and Bad is that Candidate diagnoses form a disjoint set, where any one diagnosis being present would explain the test results. Bad diagnoses do not form a disjoint set. Each Bad diagnosis is believed to be present.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

Completed : Serves as a constant for action outcomes indicating the action completed without error.

Fail : Serves as a constant for outcomes indicating a failed result.

Good : Serves as a constant for outcomes indicating a good or positive state. Specifically, the reasoner has test evidence indicating the associated diagnosis is not present.

NotAvailable : Serves as a constant for outcomes indicating the associated outcome is not available.

NotKnown : Serves as a constant for outcomes indicating the associated outcome is not known. Specifically, this corresponds to the situation where the reasoner has negligible or conflicting evidence related to this diagnosis.

NotStarted : Serves as a constant for action states (i.e., outcomes) indicating the associated action has not been initiated.

Pass : Serves as a constant for outcomes indicating a passed result.

UserDefined : Serves as a constant for outcomes indicating a user-defined result. The semantics for this constant are application/model-specific.

EXPRESS specification:

```
*)
   TYPE OutcomeValues = ENUMERATION OF
      (ABORTED,
      BAD,
      CANDIDATE,
      COMPLETED,
      FAIL,
      GOOD,
      NOTAVAILABLE,
      NOTKNOWN,
      NOTSTARTED,
      PASS,
      USERDEFINED);
   END_TYPE;
(*
```

6.1.14 ResourceCostType

Enumerated type ResourceCostType defines a type for categorizing the expense or penalty associated with the cost of an action in the diagnostic process. Current enumerated values include the following:

USER_DEFINED_COST : an unspecified application-specific cost

PERFORMANCE : the expense to execute the action

SETUP : the expense to prepare for the action

ACCESS : the expense associated with entry to the location where the action is to be performed

REENTRY : the expense to access a location a location where the action is to be performed given the same action was previously performed within the current session

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

CONSUMABL : the expense associated with replenishable material

ROTABLE : the expense associated with in-place repairs to equipment

REPAIRABLE : the expense associated with items that are repaired and returned to inventory

EXPRESS specification:

```
*)
   TYPE ResourceCostType = ENUMERATION OF
      (USER_DEFINED_COST,
      CONSUMABLE,
      ROTABLE,
      REPAIRABLE);
   END_TYPE;
(*
```

6.1.15 Role

Enumerated type Role defines the objective of an operation, mission, or test scenario.

EXPRESS specification:

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

6.1.16 SeverityCategory

Enumerated type SeverityCategory 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 SeverityCategory = ENUMERATION OF
      (CATASTROPHIC,
      CRITICAL,
      MARGINAL,
      MINOR);
   END_TYPE;
(*
```

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

6.1.17 TimeBaseline

Enumerated type TimeBaseline identifies the reference point for determining usage time. Valid values include the following:

INITIAL_OPERATION	:	indicates the time is determined relative to when the system item was first placed into service
PRIOR_INSPECTION	:	indicates the time is determined relative to when the system item last underwent an inspection or operational evaluation
PRIOR_MAINTENANCE	:	indicates the time is determined relative to when the system item last underwent either scheduled or unscheduled maintenance
PRIOR_REVISION	:	indicates the time is determined relative to when the system item last underwent a significant engineering modification
UNSPECIFIED	:	indicates the time basis is not known

EXPRESS specification:

```
*)
   TYPE TimeBaseline = ENUMERATION OF
      (INITIAL_OPERATION,
      PRIOR_INSPECTION,
      PRIOR_MAINTENANCE,
      PRIOR_REVISION,
      UNSPECIFIED);
   END_TYPE;
(*
```

6.1.18 TimeUnit

Enumerated type TimeUnits specifies units of time. The user-defined value allows for other types of time costs to be included.

EXPRESS specification:

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

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

6.1.19 FailureDistribution

Select Type FailureDistribution enables selection between a simple failure rate designation and a full probability distribution for determining probability of failure.

EXPRESS specification:

```
*)
   TYPE FailureDistribution = SELECT
      (FailureRate,
      ProbabilityDistribution);
   END_TYPE;
(*
```

6.1.20 Action

Entity Action represents a specific action taken in support of either testing or repairing an item. Actions are the primary entities to which costs are assigned. Ultimately, the diagnosis and repair processes are concerned with optimizing the sequence of actions required to return a unit to service. Test and repair are both processes that are hierarchical in nature and composed of atomic actions. This relationship provides the organizing structure for this assertion.

EXPRESS specification:

```
*)
   ENTITY Action
      SUPERTYPE OF (ONEOF(Repair, Test));
        name              : NameType;
        description       : DescriptionType;
        subAction         : OPTIONAL SET [1:?] OF Action;
        allowedStatus     : OPTIONAL LIST [2:?] OF UNIQUE ActionOutcome;
        hasCost           : OPTIONAL SET [1:?] OF Cost;
        category          : OPTIONAL ActionCostType;
        requiredResource  : OPTIONAL SET [1:?] OF Resource;
        mustOccurIn       : SET OF ContextState;
      INVERSE
        partOfModel       : DiagnosticModel FOR modelAction;
        partOf            : SET OF Action FOR subAction;
       UNIQUE
        oneName           : name;
      WHERE
        graphIsAcyclic    : NOT(EXISTS(SELF.subAction)) OR
                            actionDag(SELF,subAction);
        uniqueCostLabels  : NOT(EXISTS(hasCost)) OR
                            costLabelUnique(hasCost);
      END_ENTITY;
(*
```

Attribute definitions:

name : Attribute name provides a means for identifying the action.

description : Attribute description is used to provide an elaborated description of the action.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

subAction : Attribute subAction identifies the set of constituent actions of which this Action is composed.

allowedStatus : Attribute allowedStatus identifies the list of possible states of the action.

hasCost : Attribute hasCost identifies a set of cost entities that represents the expenses assumed to be incurred if the action is performed. This attribute is optional with a minimum cardinality of one, should it exist.

category : Attribute category specifies what type of action this is to determine the associated cost.

requiredResource : Attribute requiredResource identifies the resources required to perform a given action. This attribute is optional with a minimum cardinality of one, should it exist.

mustOccurIn : Attribute mustOccurIn identifies a disjoint set of ContextState items, where any one of them must be fulfilled for an Action to be considered. This attribute is defined to be a set. If the set is empty, then the corresponding Action is available in all ContextStates.

partOfModel : Attribute partOfModel identifies the DiagnosticModel to which the Action belongs.

partOf : Attribute partOf identifies the set of actions that have the current action as a subAction.

Formal propositions:

graphIsAcyclic Proposition graphIsAcyclic ensures that the structure of Action corresponds to a directed acyclic graph. In other words, traversing the member relationships from an Action should not result in returning to the same Action. This constraint is verified by using the function actionDag, which traverses the Action structure.

uniqueCostLabels Proposition uniqueCostLabels requires that the cost categories be unique.

6.1.21 ActionOutcome

Entity ActionOutcome defines an outcome to be associated with some action in the model. Action outcomes associate Boolean values to action states and form the basis for ordering actions in the diagnostic process.

EXPRESS specification:

```
*)
   ENTITY ActionOutcome
      SUBTYPE OF(Outcome);
      INVERSE
        forAction         : Action FOR allowedStatus;
      UNIQUE
        outcomeKey        : allowedValue,
                            forAction;
```

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

```
    WHERE
      legalValues       : SELF\Outcome.allowedValue IN
                          [NotStarted, Completed, Aborted, NotKnown,
                          NotAvailable, UserDefined];
  END_ENTITY;
(*
```

Attribute definitions:

forAction : Attribute forAction links the outcome to the specific action with that outcome.

Formal propositions:

legalValues Proposal legalValues specifies that an outcome can only have one of the associated values.

6.1.22 ContextState

Entity ContextState is a container for other user-defined conditions of a diagnostic problem, including system state, operational state, and the particular reason for performing diagnosis. Only one ContextState can be active at a given time during a diagnostic session. This influences how context is modeling in the static model.

EXPRESS specification:

```
*)
  ENTITY ContextState;
     name         :  NameType;
     description  :  DescriptionType;
     occursIn     :  SET OF ModeOfOperation;
     hasPurpose   :  Purpose;
    UNIQUE
     oneName  :   name;
  END_ENTITY;
(*
```

Attribute definitions:

name : Attribute name provides a unique identifier for a particular ContextState within the model.

description : Attribute description provides a means of associating descriptive text with the ContextState.

occursIn : Attribute occursIn identifies the relevant operational mode within which the particular maintenance actions are being performed.

hasPurpose : Attribute hasPurpose identifies the purpose of actions performed in the given ContextState.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

6.1.23 Cost

Entity Cost specifies the expense or penalty that is expected to be incurred relative to the performance of some action. It is typically used as a metric for optimization via some objective function. Costs are categorized by the type of cost to which they relate. One dimension of Cost identifies whether the cost is a measure of time or some other unit. The second dimension to Cost is based on the task to which the cost pertains: for time costs, performance, setup, access, reentry, and for non-time costs, consumable, rotable, and repairable. Each cost has an associated unit to enable consistent processing of included values.

EXPRESS specification:

```
*)
  ENTITY Cost
    ABSTRACT SUPERTYPE OF (ONEOF(TimeCost, NonTimeCost));
       upperBound       : OPTIONAL CostValue;
       lowerBound       : OPTIONAL CostValue;
       predictedValue   : OPTIONAL CostValue;
       costLabel        : CostCategory;
    WHERE
       validBound       : NOT(EXISTS(upperBound)) OR
                          NOT(EXISTS(lowerBound)) OR
                          (lowerBound <= upperBound);
       validUpperBound  : NOT(EXISTS(predictedValue)) OR
                          NOT(EXISTS(upperBound)) OR
                          (predictedValue <= upperBound);
       validLowerBound  : NOT(EXISTS(predictedValue)) OR
                          NOT(EXISTS(lowerBound)) OR
                          (lowerBound <= predictedValue);
       boundOrValue     : (EXISTS(predictedValue)) OR
                          (EXISTS(upperBound) AND EXISTS(lowerBound));
  END_ENTITY;
(*
```

Attribute definitions:

upperBound : Attribute upperBound provides the nominal upper limit for the value of the cost entity.

lowerBound : Attribute lowerBound provides the nominal lower limit for the value of the cost entity.

predictedValue : Attribute predictedValue provides the nominal, expected, or some other predicted value for the cost entity.

costLabel : Attribute costLabel provides a name and description for an implementation to label like costs to be used for optimizing the test or repair process (e.g., all of the costs associated with the skill level required to perform a test).

Formal propositions:

validBound : When they exist, proposition validBound ensures the lowerBound and upperBound attributes are constrained to values such that the value of the lowerBound attribute is less than or equal to the upperBound attribute.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

validUpperBound — When they exist, proposition validUpperBound ensures the predictedValue and upperBound attributes are constrained to values such that the value of the predictedValue attribute is less than or equal to the upperBound attribute.

validLowerBound — When they exist, proposition validLowerBound ensures the predictedValue and lowerBound attributes are constrained to values such that the value of the lowerBound attribute is less than or equal to the predictedValue attribute.

boundOrValue — Proposition boundOrValue requires that either a predictedValue be provided or both upper and lower bounds be provided. Because the constraint uses an inclusive-OR, all three may be provided.

6.1.24 CostCategory

Entity CostCategory defines the type or category of cost element being defined for the model. This entity provides identifying information (name and description) for the cost metric. This entity permits identification of a class of cost metrics to be used when determining optimization criteria for the reasoner to follow. This is used with an entity in the DCM to select the cost-based optimization criteria.

EXPRESS specification:

```
*)
  ENTITY CostCategory;
       name           : NameType;
       description    : DescriptionType;
    INVERSE
       specifiedCost  : SET [1:?] OF Cost FOR costLabel;
    UNIQUE
       oneName :      name;
  END_ENTITY;
(*
```

Attribute definitions:

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

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

specifiedCost : Attribute specifiedCost identifies the set of costs defined within a diagnostic model belonging to the give category (e.g., all of the skill-level costs).

6.1.25 Diagnosis

Entity Diagnosis corresponds to a conclusion or group of conclusions to be drawn about a system or unit under test. Diagnosis is a directed acyclic graph (not just a tree); therefore, any particular diagnosis can belong to one or more higher level groups of diagnoses (e.g., belonging to one or more higher order functions) and contain one or more lower level diagnoses (e.g., containing multiple subfunctions). At run time, the outcome of a particular diagnosis is logically consistent as a roll up of the outcomes of its child diagnoses such that if the parent diagnosis is good, then all child diagnoses must also be good. Similarly, if any child diagnosis is a candidate, then so is the parent diagnosis. Interoperability is not assured in the presence of user-defined outcomes without prior agreement on the associated inference rules.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

A Diagnosis that is neither a Fault nor a Failure and has no children indicates that the Diagnosis has not been modeled to the level of detail of a Fault or Failure. It simple represents a conclusion that can be drawn.

Some forms of diagnostic reasoning require the specification of an explicit NoFault diagnosis entity; however, others would not work properly if such an entity were included. If the model includes a diagnosis for NoFault, then that model shall use the "NoFault" constant in the name of that entity.

EXPRESS specification:

```
*)
  ENTITY Diagnosis
    SUPERTYPE OF (ONEOF(Failure, Fault));
      name                : NameType;
      description         : DescriptionType;
      subGroup            : OPTIONAL SET [1:?] OF Diagnosis;
      allowedOutcomes     : OPTIONAL LIST [2:?] OF UNIQUE
                            DiagnosisOutcome;
      hasDistribution     : OPTIONAL FailureDistribution;
      severity            : OPTIONAL SeverityCategory;
      atIndentureLevel    : OPTIONAL Level;
      mustOccurIn         : SET OF ContextState;
    INVERSE
      partOfModel         : DiagnosticModel FOR modelDiagnosis;
      memberOf            : SET OF Diagnosis FOR subGroup;
    UNIQUE
      oneName :           name;
    WHERE
      outcomesRequiredForAtomicDiagnosis  : (EXISTS(SELF.subGroup)) OR
                                 EXISTS(allowedOutcomes);
      minimalOutcomes    :  (NOT(EXISTS(allowedOutcomes))) OR
                            ((SIZEOF(QUERY(tmp <* allowedOutcomes |
                                 tmp.allowedValue = Good)) = 1) AND
                             (SIZEOF(QUERY(tmp <* allowedOutcomes |
                                 tmp.allowedValue = Candidate)) = 1));
      allOrNone          :  NOT(EXISTS(SELF.subGroup)) OR
                            ((SIZEOF(QUERY(tmp <* subGroup |
                            EXISTS(hasDistribution))) =
                            SIZEOF(subGroup)) XOR
                             (SIZEOF(QUERY(tmp <* subGroup |
                            EXISTS(hasDistribution))) = 0));
      graphIsAcyclic     :  NOT(EXISTS(SELF.subGroup)) OR
                            diagnosisDag(SELF,subGroup);
      parentLevelConsistent   :  ((NOT(EXISTS(subGroup)) OR
                                 (SIZEOF(subGroup) = 0)) OR
                                  (NOT(EXISTS(atIndentureLevel)) AND
                                   (SIZEOF(QUERY(tmp <* subGroup |
                                 EXISTS(tmp.atIndentureLevel))) = 0))
                                 OR
                                  (EXISTS(atIndentureLevel) AND
                                   (SIZEOF(QUERY(tmp <* subGroup |
                                 NOT(EXISTS(tmp.atIndentureLevel)))) =
                                 0) AND
                                   (SIZEOF(QUERY(tmp <* subGroup |
                                 (SELF.atIndentureLevel =
```

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62243 2012

```
                                    tmp.atIndentureLevel) OR
                                    (EXISTS(SELF.atIndentureLevel.successo
                                    r) AND
                                    (SELF.atIndentureLevel.successor =
                                    tmp.atIndentureLevel)))) =
                                       SIZEOF(SELF.subGroup))));
      forNoFault          :  NOT(SELF.name = NoFault) OR
                             ((SIZEOF(SELF.subGroup) = 0) AND
                              (SIZEOF(SELF.memberOf) = 0) AND
                              (NOT('AI_ESTATE_CEM.FAULT' IN
                             typeof(SELF))) AND
                              (NOT('AI_ESTATE_CEM.FAILURE' IN
                             typeof(SELF))));
  END_ENTITY;
(*
```

Attribute definitions:

name : Attribute name is used to identify the particular Diagnosis uniquely.

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

subGroup : Attribute subGroup identifies the set of constituent diagnoses of which this Diagnosis is a logically consistent roll up.

allowedOutcomes : Attribute allowedOutcomes provides a list of two or more allowable outcomes of the diagnosis. It is a list because some specific diagnostic models require an order to be imposed on the available outcomes. This attribute is shown to be optional; however, it is constrained such that it is required if the diagnosis has no member diagnoses.

hasDistribution : Attribute hasDistribution provides the failure distribution associated with the given diagnosis. Failure distributions at any hierarchical level above the lowest, if they exist, represent user-implemented aggregates of subGroup failure distributions.

severity : Attribute severity associates a level of severity for the given diagnosis. This attribute is optional because 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.

atIndentureLevel : Attribute atIndentureLevel identifies the specific level of indenture for a particular Diagnosis.

mustOccurIn : Attribute mustOccurIn provides a disjoint set of ContextState items, where any one of them must be fulfilled for an Diagnosis to be considered. This attribute is defined to be a set. If the set is empty, then the corresponding Diagnosis is available in all ContextStates.

partOfModel : Attribute partOfModel identifies the DiagnosticModel to which the Diagnosis belongs.

memberOf : Attribute memberOf identifies the set of diagnoses that have the current Diagnosis as a parent group.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

Formal propositions:

outcomesRequiredForAtomicDiagnosis	Proposition outcomesRequiredForAtomicDiagnosis 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 subdiagnoses) have outcomes. The cardinality on the DiagnosisOutcome 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.
minimalOutcomes	Proposition minimalOutcomes 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.
allOrNone	Proposition allOrNone ensures that either all of the children have a failure distribution associated or none of the children have a failure distribution associated.
graphIsAcyclic	Proposition graphIsAcyclic ensures the structure of Diagnosis 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 diagnosisDag, which traverses the Diagnosis structure.
parentLevelConsistent	Proposition parentLevelConsistent ensures that one of the following shall be true: 1) there are no child Diagnoses, 2) no indenture levels exist for this Diagnosis and its children, or 3) the indenture levels are consistent for this Diagnosis and its children. Specifically, the constraint is imposed that the level of the child Diagnosis shall be the same as the current Diagnosis or shall be successors to the current level.
forNoFault	Proposition forNoFault requires that if a NoFault diagnosis is present, then that diagnosis has no parents, no children, is not a fault, and is not a failure.

6.1.26 DiagnosisOutcome

Entity DiagnosisOutcome defines one possible discrete outcome associated with a diagnosis entity in the model. For each diagnosis, a set of three outcomes are defined for use by the diagnostic reasoner. More diagnosisOutcomes can be associated with a diagnosis if user-defined extensions are required (e.g., adding suspect or bad).

EXPRESS specification:

```
*)
  ENTITY DiagnosisOutcome
    SUBTYPE OF(Outcome);
    INVERSE
      forDiagnosis  : Diagnosis FOR allowedOutcomes;
```

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

```
    UNIQUE
      outcomeKey    : allowedValue,
                      forDiagnosis;
    WHERE
      legalValues   : SELF\Outcome.allowedValue IN
                      [Good, Candidate, Bad, NotKnown, UserDefined];
  END_ENTITY;
(*
```

Attribute definitions:

forDiagnosis : Attribute forDiagnosis links the outcome to the specific diagnosis with that outcome.

Formal propositions:

legalValues Proposition legalValues specifies that an outcome can only have one of the associated values, where the semantics for the values are defined as follows:

GOOD : The reasoner has test evidence (i.e., at least one test outcome supporting the diagnosis) indicating the associated diagnosis is not present.

BAD : The reasoner has test evidence indicating the associated diagnosis is present.

CANDIDATE : The reasoner has test evidence indicating the diagnosis may be present, although the test evidence is also consistent with either some other diagnosis being present instead of this one or a conjoint set of multiple diagnoses being present that does not include this one.

NOT_KNOWN : The reasoner has negligible test evidence related to this diagnosis.

USER_DEFINED : Unspecified.

6.1.27 DiagnosticModel

Entity DiagnosticModel provides a container of the elements that provide information for diagnosing a system under test. Diagnostic reasoning involves drawing conclusions from test outcomes. The relationships between test outcomes and diagnoses are defined as subtypes of this entity in related schemas within this standard.

EXPRESS specification:

```
*)
  ENTITY DiagnosticModel;
    name              : NameType;
    description       : DescriptionType;
    modelItem         : SET [1:?] OF SystemItem;
```

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

```
    modelAction       : SET [1:?] OF Action;
    modelDiagnosis    : SET [2:?] OF Diagnosis;
    systemUnderTest   : SystemItem;
  UNIQUE
    oneName   :       name;
  WHERE
    elementIsRollup   : itemRollup(SELF,SELF.modelItem) AND
                        actionRollup(SELF,SELF.modelAction) AND
                        diagnosisRollup(SELF,SELF.modelDiagnosis);
    atLeastOneTest    : (SIZEOF(QUERY(tmp <* modelAction |
                        'AI_ESTATE_CEM.TEST' in TYPEOF(tmp))) >= 1);
    systemInModel     : (systemUnderTest IN modelItem);
    noParent  :       SIZEOF(SELF.systemUnderTest.partOf) = 0;
  END_ENTITY;
(*
```

Attribute definitions:

name	:	Attribute name provides a means for identifying the DiagnosticModel.
description	:	Attribute description is used to provide an elaborated description of the DiagnosticModel.
modelItem	:	Attribute modelItem identifies the various SystemItems in the diagnostic model.
modelAction	:	Attribute modelAction identifies the various actions in a diagnostic model. At least one action must exist corresponding to at least one Test.
modelDiagnosis	:	Attribute modelDiagnosis identifies the various diagnoses in a diagnostic model. At least two diagnoses must exist; otherwise, diagnosis is trivial.
systemUnderTest	:	Attribute systemUnderTest identifies the top-level SystemItem that is the object of diagnosis for the DiagnosticModel.

Formal propositions:

elementIsRollup	Proposition elementIsRollup verifies that all of the SystemItem, Action, and Diagnosis entities referenced at this level are defined as being part of this model.
atLeastOneTest	Proposition atLeastOneTest ensures that a diagnostic model has at least one test entity.
systemInModel	Proposition systemInModel ensures that the systemUnderTest is listed as a SystemItem that is “partOf” the model.
noParent	Proposition noParent ensures that the systemUnderTest for the DiagnosticModel has no parent SystemItems.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

6.1.28 Failure

Entity Failure is a subtype of Diagnosis tied to a FunctionItem and corresponds to 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);
       failedItem     :  OPTIONAL FunctionItem;
  INVERSE
       physicalCause  :  SET OF Fault FOR causedFailure;
  END_ENTITY;
(*
```

Attribute definitions:

failedItem : Attribute failedItem identifies the specific FunctionItem that has failed.

physicalCause : Inverse attribute physicalCause identifies the physical fault leading to the given failure. The minimum cardinality of zero indicates that the cause–effect relationship between fault and failure may not be known.

6.1.29 FailureMechanism

Entity FailureMechanism is used to describe the underlying first principle cause for a fault or failure. It is associated first with the diagnosis then (by way of the appropriate diagnosis subtype) to the RepairItem or FunctionItem that has failed.

EXPRESS specification:

```
*)
  ENTITY FailureMechanism;
    name          : NameType;
    description   : DescriptionType;
  UNIQUE
    oneName   :   name;
  END_ENTITY;
(*
```

Attribute definitions:

name : Attribute name is used to identify uniquely the FailureMechanism.

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

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

6.1.30 Fault

Entity Fault is a subtype of Diagnosis associated with a RepairItem and corresponds to a physical cause of anomalous behavior within a system.

EXPRESS specification:

```
*)
   ENTITY Fault
     SUBTYPE OF(Diagnosis);
     causedFailure     : OPTIONAL SET OF Failure;
     failedItem        : OPTIONAL RepairItem;
     rootCause         : OPTIONAL SET [1:?] OF FailureMechanism;
   WHERE
     faultsAtRepairItemLeaf   : NOT(EXISTS(failedItem)) OR
                                NOT(EXISTS(failedItem.subassembly));
     faultAtFailureLeaf       : NOT(EXISTS(causedFailure)) OR
                                (SIZEOF(causedFailure) = 0) OR
                                (SIZEOF(QUERY(tmp <* causedFailure |
                                  EXISTS(tmp.subGroup)))=0);
     faultAtLeaf              : NOT(EXISTS(subGroup));
   END_ENTITY;
(*
```

Attribute definitions:

causedFailure : Attribute causedFailure identifies zero or more associated functional failures caused by the presence of the given physical fault. The minimum cardinality of zero indicates that the failure may not be observable or that the cause–effect relationship may not be known.

failedItem : Attribute failedItem identifies the specific RepairItem that contains the associated Fault.

rootCause : Attribute rootCause identifies the first principle failure mechanisms associated with this diagnosis.

Formal propositions:

faultsAtRepairItemLeaf — Proposition faultsAtRepairItemLeaf requires that faults occur only at the bottom of the repair item hierarchy.

faultAtFailureLeaf — Proposition faultAtFailureLeaf ensures that the caused failure is at the bottom of the failure hierarchy.

faultAtLeaf — Proposition faultAtLeaf ensures that faults only occur at the leaves of the diagnosis directed acyclic graph.

6.1.31 FunctionItem

Entity FunctionItem represents a node in a functional/behavioral decomposition of a system. The parent/child relationships of FunctionItem should represent the decomposition hierarchy.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

EXPRESS specification:

```
*)
  ENTITY FunctionItem
    SUBTYPE OF(SystemItem);
      SELF\SystemItem.subAssembly    : OPTIONAL SET [1:?] OF
                                       FunctionItem;
  INVERSE
    implementedBy   :  SET OF RepairItem FOR includesFunction;
    failsAs         :  SET [1:?] OF Failure FOR failedItem;
  END_ENTITY;
(*
```

Attribute definitions:

subAssembly : Attribute subAssembly redeclares the inherited SELF\SystemItem.subAssembly attribute to ensure that children are of the same type (i.e., children are also FailureItems).

implementedBy : Inverse attribute implementedBy identifies the repair item or items used to implement a particular function within a system.

failsAs : Inverse attribute failsAs identifies the specific set of Diagnoses for the SystemItem.

6.1.32 Level

Entity Level provides a mechanism for grouping Actions, Diagnoses, and SystemItems at some common slice of the hierarchy.The specification of levels is model-specific except that a level is constrained to be related to at most one predecessor level and at most one successor level in a total order.

EXPRESS specification:

```
*)
  ENTITY Level;
    name          :  NameType;
    description   :  DescriptionType;
    successor     :  OPTIONAL Level;
  INVERSE
    predecessor   :  SET [0:1] OF Level FOR successor;
  UNIQUE
    oneName   :   name;
  WHERE
    noRepeats :   levelsAcyclic(SELF);
  END_ENTITY;
(*
```

Attribute definitions:

name : Attribute name specifies a unique name for the Level of the SystemItem and Diagnosis entities. This attribute is intended to provide a means of identifying Levels for purposes of determining applicability of tests, diagnoses, etc. within the model.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

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

successor : Attribute successor identifies the next Level in the total order of Levels. This is optional in that the final Level will not have a successor.

predecessor : Inverse attribute predecessor identifies the previous Level in the total order of Levels when the previous Level exists.

Formal propositions:

noRepeats Proposition noRepeats tests to see whether the current entity has the same Level appearing in the successor chain. If so, it creates a cycle in the Levels, which is illegal.

6.1.33 ModeOfOperation

Entity ModeOfOperation is a named representation of system state or operational state that contributes to some ContextState that is relevant to the diagnostic process.

EXPRESS specification:

```
*)
   ENTITY ModeOfOperation;
      name          : NameType;
      description   : DescriptionType;
   UNIQUE
      oneMode   :   name;
   END_ENTITY;
(*
```

Attribute definitions:

name : Attribute name is a unique attribute used to identify the mode of operation.

description : Attribute description provides a textual description of the specific mode of operation within which the system is being diagnosed.

6.1.34 NonTimeCost

Entity NonTimeCost is an expense that is represented in terms of currency or some nontemporal metric that supports an objective function.

EXPRESS specification:

```
*)
   ENTITY NonTimeCost
      SUBTYPE OF(Cost);
         nonTimeCostUnit    :  NonTimeUnit;
         costRate           :  OPTIONAL Rate;
   END_ENTITY;
(*
```

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

Attribute definitions:

nonTimeCostUnit : Attribute nonTimeCostUnit defines the non-time units associated with the non–time-related cost.

costRate : Attribute costRate provides a rate of expenditure for the specific non-time cost. In other words, this attribute will provide the cost unit part of cost-per-unit-time. If this attribute is present, this entity represents a cost rate. If absent, it represents a fixed cost.

6.1.35 Outcome

Entity Outcome represents a legal discrete value to be associated with tests, diagnoses, or actions. These values form the basis for directing the diagnostic process.

EXPRESS specification:

```
*)
  ENTITY Outcome
    ABSTRACT SUPERTYPE OF (ONEOF(DiagnosisOutcome, TestOutcome,
                            ActionOutcome));
      maxConfidence    :  OPTIONAL ConfidenceValue;
      allowedValue     :  OutcomeValues;
  END_ENTITY;
(*
```

Attribute definitions:

maxConfidence : Attribute maxConfidence sets an inclusive upper bound on actual confidence values for the outcome that can be observed or inferred during a session. The upper bound constraint is in addition to constraints built into the type used for actual confidence values: ConfidenceValue. The value of the bound is based on the domain-specific characteristics of the outcome.

allowedValue : Attribute allowedValue defines a value that can belong to the specific outcome based on the subtype instantiating the Outcome entity.

6.1.36 ProbabilityDistribution

Entity ProbabilityDistribution defines the key parameters for a generalized failure distribution. The intent is to provide the means for representing the shape of any distribution and is constructed using a group of synchronized lists. The synchronization is achieved using the timeStep attribute that specifies a relative delta-t from the start of the sequence. The distribution definition includes a probability distribution function (PDF) and a cumulative distribution function (CDF); however, the CDF can be derived from the PDF. To support variable failure rates, a list of failure rates can be specified as well.

EXPRESS specification:

```
*)
  ENTITY ProbabilityDistribution;
    cdf                  : LIST [2:?] OF ProbabilityValue;
    timeStep             : LIST [2:?] OF TimeValue;
```

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

```
    basis                 : TimeBaseline;
    usageUnit             : TimeUnit;
    pdf                   : LIST [2:?] OF DistributionPoint;
  WHERE
    synchronizedLists     : (SIZEOF(timeStep)=SIZEOF(pdf))
                            AND (SIZEOF(timeStep)=SIZEOF(cdf));
    validStartTime        : (timeStep[1]=0.0);
    validStartCDF         : (cdf[1]=1.0);
    increasingTime        : increasingTimeCheck(timeStep);
    decreasingCDF         : decreasingCdfCheck(cdf);
    validCDFIntegration   : cdfIntCheck(timeStep,pdf,cdf);
  END_ENTITY;
(*
```

Attribute definitions:

cdf : Attribute cdf provides a list of values corresponding to an approximation of one minus the cumulative density function for the associated distribution.

timeStep : Attribute timeStep identifies the point in time at which the corresponding distribution value is associated.

basis : Attribute basis provides the basis or foundation upon which the usage time is determined.

usageUnit : Attribute usageUnit specifies the units for the time step of the probability distribution. Specifically, the units are per usageUnit.

pdf : Attribute pdf defines a list of values in order of increasing time providing an approximation of the probability density function for the associated failure distribution.

Formal propositions:

synchronizedLists	Proposition synchronizedLists verifies that cdf, pdf, failureRate, and timeStep lists are of equal size.
validStartTime	Proposition validStartTime verifies that the first value in the timeStep list is 0.
validStartCDF	Proposition validStartCDF verifies that the first value in the cdf list is 1.
increasingTime	Proposition increasingTime verifies that the timeStep list is monotonically increasing.
decreasingCDF	Proposition decreasingCDF verifies that the cdf list is monotonically nonincreasing.
validCDFIntegration	Proposition validCDFIntegration verifies that the cdf list is plausibly an integral of the pdf.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

6.1.37 Purpose

Entity Purpose specifies the reason for an associated action.

EXPRESS specification:

```
*)
  ENTITY Purpose;
    hasRole       :  SET [1:?] OF Role;
    description   :  DescriptionType;
  END_ENTITY;
(*
```

Attribute definitions:

hasRole : Attribute hasRole identifies the associated role of the ContextState within which test/diagnosis is occurring. For example, the role may be a maintenance test or a verification test.

description : Attribute description provides a textual description of the purpose of the ContextState.

6.1.38 Rate

Entity Rate defines a ratio of notTimeCost to some timeUnit. It can be used, for example, to specify labor rates, rental rates, or other costs per time unit (e.g., dollars per 100 h).

EXPRESS specification:

```
*)
  ENTITY Rate;
    timeCostUnit    : TimeUnit;
    unitMultiplier  : CostValue;
  END_ENTITY;
(*
```

Attribute definitions:

timeCostUnit : Attribute timeCostUnit provides the time units for computing the cost rate. In other words, this attribute will provide the time part of cost-per-unit-time.

unitMultiplier : Attribute unitMultiplier provides a divisor for the Rate, thus permitting values other than unity.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

6.1.39 Repair

Entity Repair is an action required to restore a RepairItem.

EXPRESS specification:

```
*)
   ENTITY Repair
      SUBTYPE OF(Action);
      SELF\Action.subAction   : OPTIONAL SET [1:?] OF Repair;
   END_ENTITY;
(*
```

Attribute definitions:

subAction : Attribute subAction redeclares the inherited SELF\Action.subAction attribute to ensure that children are of the same type (i.e., children are also Repairs).

6.1.40 RepairItem

Entity RepairItem refers to a part of the system under test on which a maintenance action can be performed as the result of a diagnosis.

EXPRESS specification:

```
*)
   ENTITY RepairItem
      SUBTYPE OF(SystemItem);
      includesFunction             :  OPTIONAL SET [1:?] OF FunctionItem;
      repairedBy                   :  SET OF Repair;
      SELF\SystemItem.subAssembly  :  OPTIONAL SET [1:?] OF RepairItem;
   INVERSE
      failsAs                      :  SET [1:?] OF Fault FOR failedItem;
   END_ENTITY;
(*
```

Attribute definitions:

includesFunction : Attribute includesFunction identifies the FunctionItems that the RepairItem plays a role in implementing. Each RepairItem may implement multiple FunctionItems, and multiple RepairItems may play a role in implementing a FunctionItem.

repairedBy : Attribute repairedBy identifies a (possibly empty) set of repairs available to repair a given repair item.

subAssembly : Attribute subAssembly redeclares the inherited SELF\SystemItem.subAssembly attribute to ensure that children are of the same type (i.e., children are also RepairItems).

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

failsAs : Inverse attribute failsAs identifies the set of specific Faults for the RepairItem. A rule is defined on Fault to forbid RepairItems anywhere but at leaves.

6.1.41 Resource

Entity Resource refers to an asset required to perform some action in the maintenance process (e.g., a piece of equipment or personnel).

EXPRESS specification:

```
*)
  ENTITY Resource;
    name       :  NameType;
    hasCost    :  OPTIONAL SET [1:?] OF Cost;
    category   :  ResourceCostType;
  UNIQUE
    oneName   :   name;
  END_ENTITY;
(*
```

Attribute definitions:

name : Attribute name provides a unique, identifying name for the resources.

hasCost : Attribute hasCost associates a set of costs with this particular resource.

category : Attribute category specifies what type of resource this is to determine the associated cost.

6.1.42 SystemItem

Entity SystemItem is an abstract supertype corresponding to a system RepairItem or FunctionItem that is the object of the diagnostic process.

EXPRESS specification:

```
*)
  ENTITY SystemItem
    ABSTRACT SUPERTYPE OF (ONEOF(RepairItem, FunctionItem));
    name               : NameType;
    description        : DescriptionType;
    subAssembly        : OPTIONAL SET [1:?] OF SystemItem;
    hasDistribution    : OPTIONAL FailureDistribution;
    atIndentureLevel   : OPTIONAL Level;
    mustOccurIn        : SET OF ContextState;
    testedBy           : SET OF Test;
  INVERSE
    modelForSystem   :  SET [0:1] OF DiagnosticModel FOR
                        systemUnderTest;
    partOfModel      :  DiagnosticModel FOR modelItem;
    partOf           :  SET OF SystemItem FOR subAssembly;
  UNIQUE
```

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

```
    oneName   :      name;
  WHERE
    allOrNone :      NOT(EXISTS(SELF.subassembly)) OR
                        ((SIZEOF(QUERY(tmp <* subAssembly |
                        EXISTS(hasDistribution))) = SIZEOF(subAssembly))
                        XOR
                         (SIZEOF(QUERY(tmp <* subAssembly |
                        EXISTS(hasDistribution))) = 0));
    graphIsAcyclic   :  NOT(EXISTS(SELF.subAssembly)) OR
                        itemDag(SELF,subAssembly);
    parentLevelConsistent : ((NOT(EXISTS(subAssembly)) OR
                            (SIZEOF(subAssembly) = 0)) OR
                             (NOT(EXISTS(atIndentureLevel)) AND
                              (SIZEOF(QUERY(tmp <* subAssembly |
                            EXISTS(tmp.atIndentureLevel))) = 0)) OR
                             (EXISTS(atIndentureLevel) AND
                              (SIZEOF(QUERY(tmp <* subAssembly |
                            NOT(EXISTS(tmp.atIndentureLevel)))) = 0)
                            AND
                              (SIZEOF(QUERY(tmp <* subAssembly |
                                           (SELF.atIndentureLevel =
                            tmp.atIndentureLevel) OR
                            (EXISTS(SELF.atIndentureLevel.successor)
                            AND (SELF.atIndentureLevel.successor =
                            tmp.atIndentureLevel)))) =
                               SIZEOF(SELF.subAssembly))));
  END_ENTITY;
(*
```

Attribute definitions:

name	:	Attribute name is used to uniquely identify the particular SystemItem.
description	:	Attribute description is used to provide an elaborated explanation of the SystemItem.
subAssembly	:	Attribute subAssembly identifies the set of constituent SystemItems of which this SystemItem is composed.
hasDistribution	:	Attribute hasDistribution identifies the specific failure distribution for the system item. Failure distributions at any hierarchical level above the lowest, if they exist, represent user-implemented aggregates of subAssembly failure distributions.
atIndentureLevel	:	Attribute atIndentureLevel identifies the specific level of indenture for a particular SystemItem.
mustOccurIn	:	Attribute mustOccurIn provides a disjoint set of ContextState items, where any one of them must be fulfilled for an SystemItem to be considered. This attribute is defined to be a set. If the set is empty, then the corresponding SystemItem is available in all ContextStates.
testedBy	:	Attribute testedBy identifies a set (possibly empty) of tests available to test a particular repair item.

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012

modelForSystem : Inverse attribute modelForSystem identifies the diagnostic model representing the specific systemUnderTest.

partOfModel : Inverse attribute partOfModel identifies the DiagnosticModel to which the SystemItem belongs.

partOf : Inverse attribute partOf identifies the set of SystemItems that have the current SystemItem as a subAssembly.

Formal propositions:

allOrNone — Proposition allOrNone ensures that either all of the children have a failure distribution associated or none of the children have a failure distribution associated.

graphIsAcyclic — Proposition graphIsAcyclic ensures the structure of SystemItem corresponds to a directed acyclic graph. In other words, traversing the member relationships from a SystemItem should not result in returning to the same SystemItem. This constraint is verified by using the function itemDag, which traverses the SystemItem structure.

parentLevelConsistent — Proposition parentLevelConsistent ensures that one of the following shall be true: 1) there are no child SystemItems, 2) no indenture levels exist for this SystemItem and its children, or 3) the indenture levels are consistent for this SystemItems and its children. Specifically, the constraint is imposed that the level of the child SystemItem shall be the same as the current SystemItem or shall be successors to the current level.

6.1.43 Test

Entity Test is a discrete information source whose output is one of a set of enumerated values that can indicate the health of some portion of the system.

EXPRESS specification:

```
*)
   ENTITY Test
     SUBTYPE OF(Action);
       allowedOutcomes         :  OPTIONAL LIST [2:?] OF UNIQUE
                                  TestOutcome;
       SELF\Action.subAction   :  OPTIONAL SET [1:?] OF Test;
   WHERE
     outcomesRequiredForAtomicTest   : (SIZEOF(SELF\Action.subAction) >
                                       0) OR EXISTS(allowedOutcomes);
     minimalOutcomes  : (NOT(EXISTS(allowedOutcomes))) XOR
                        ((SIZEOF(QUERY(tmp <* allowedOutcomes |
                         tmp.allowedValue = Pass)) = 1)
                       AND(SIZEOF(QUERY(tmp <* allowedOutcomes |
                         tmp.allowedValue = Fail)) >= 1));
     uniqueUnqualifiedOutcomes   : (NOT(EXISTS(allowedOutcomes))) XOR
                                   ((SIZEOF(QUERY(tmp <* allowedOutcomes
                                   |
                                    (tmp.allowedValue = FAIL) AND
                                    (NOT(EXISTS(tmp.qualifier))))) <=
```

Published by IEC under license from IEEE. © 2010 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: IEC 62243 2012