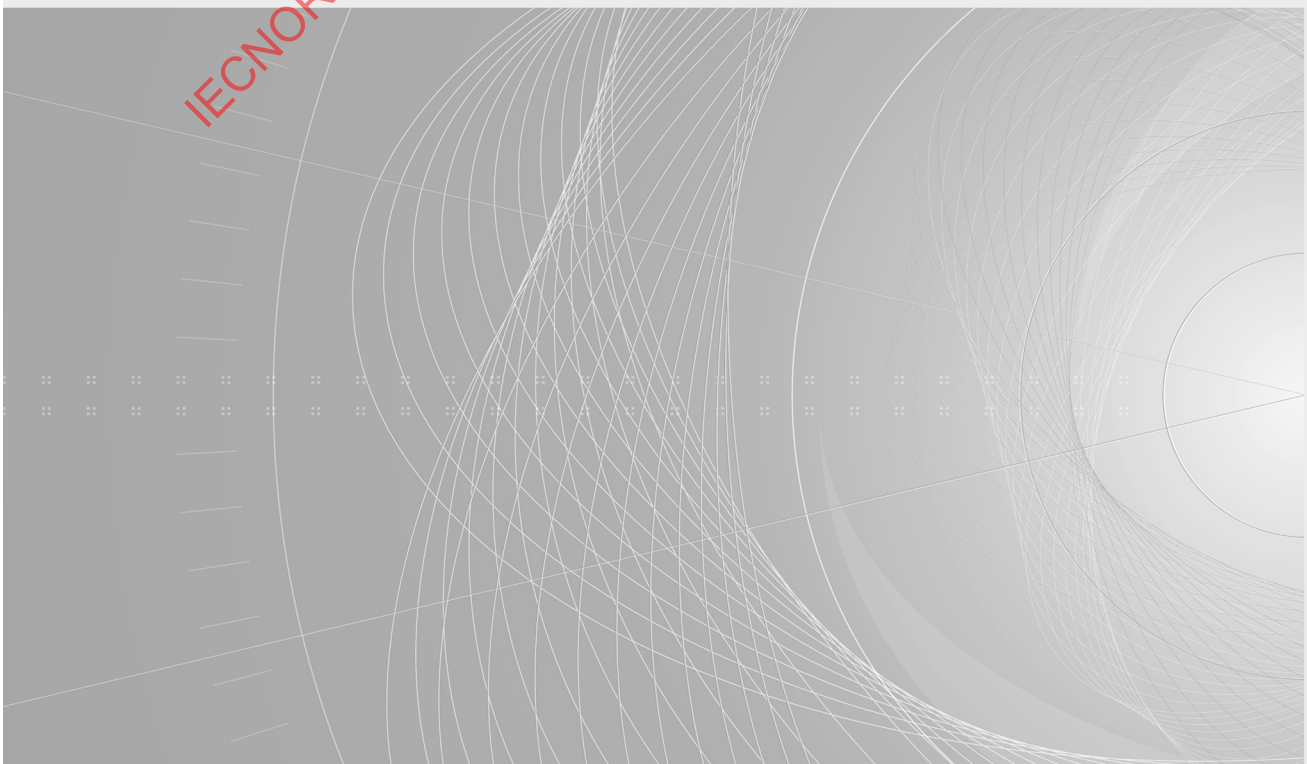
IEC 62243

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

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

IEC 62243

Edition 2.0 2012-06

IEEE Std 1232™

INTERNATIONAL STANDARD

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

Contents

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.

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.

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

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 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.

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.

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.

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.

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.

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.

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/.

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).

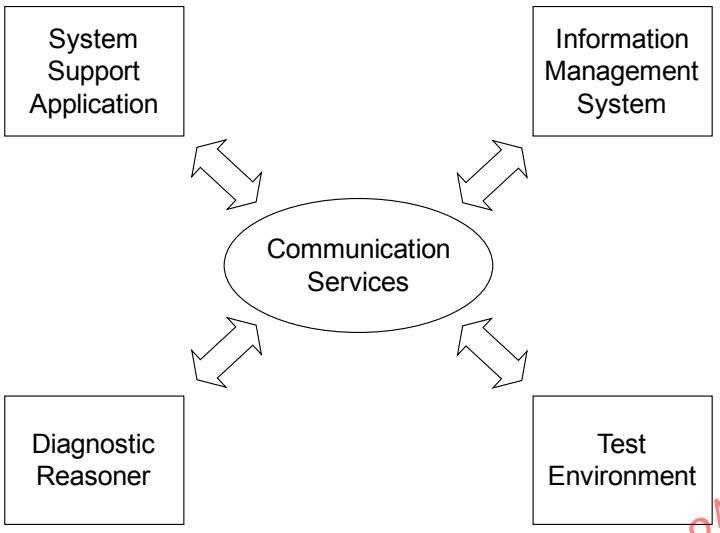


Figure 1—AI-ESTATE architectural concept

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.

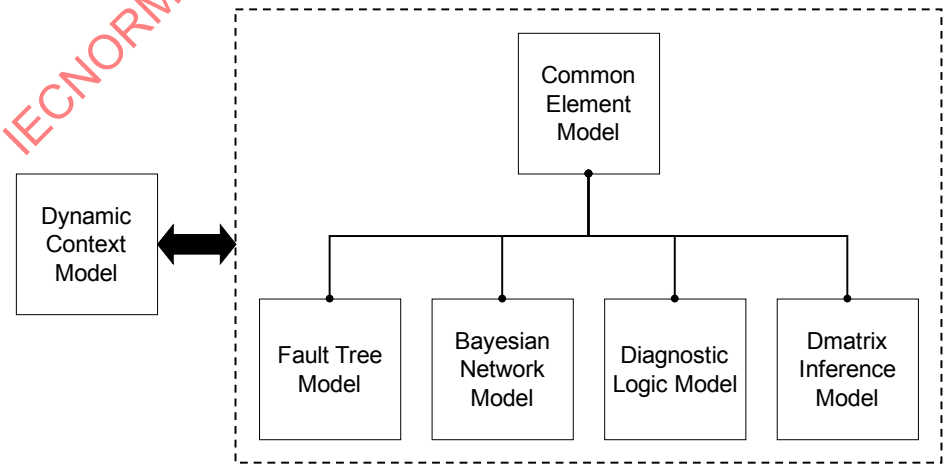


Figure 2—Hierarchical structure of the AI-ESTATE models

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.

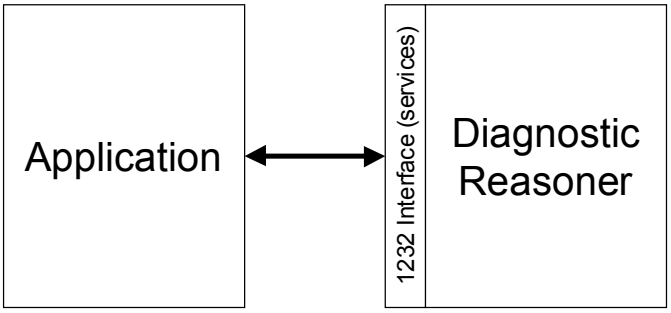


Figure 3—AI-ESTATE interface layer

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);
```

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.

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.

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.

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.

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.

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 ProbabilityValue 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;
(*
```

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

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.

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

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;
(*
```

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;
(*
```

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.

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;
```

```
    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.

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.

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.

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 =
```

```
                                    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.

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;
```

```
    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;
```

```
    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.

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.

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.

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.

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;
(*
```

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;
```

```
    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.

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.

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).

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
```

```
    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.

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))))) <=
```

```
                                        1));
  END_ENTITY;
(*
```

Attribute definitions:

allowedOutcomes	:	Attribute allowedOutcomes provides a list of two or more test outcomes that are the expected outcomes of the test. A test outcome is a characterization of the observed response to the stimulus of a test. This attribute is shown to be optional; however, it is constrained such that it is required if the test has no member tests.
subAction	:	Attribute subAction redeclares the inherited SELF\Action.subAction attribute to ensure that children are of the same type (i.e., children are also Tests).

Formal propositions:

outcomesRequiredForAtomicTest	Proposition outcomesRequiredForAtomicTest determines whether TestOutcomes are associated with a test and requires that an atomic test (i.e., a test for which there are no subtests) have TestOutcomes. The cardinality on the TestOutcome set ensures that, should TestOutcomes exist, there are at least two of them. Note that nonatomic tests are permitted, but not required, to have TestOutcomes.
minimalOutcomes	Proposition minimalOutcomes requires either that the set of TestOutcomes not be defined or that the set of TestOutcomes include at a minimum, exactly one outcome of value PASS and at least one outcome of value FAIL.
uniqueUnqualifiedOutcomes	Proposition uniqueUnqualifiedOutcomes requires that if the qualifier is present, then only one FAIL TestOutcome with a given qualifier can be used.

6.1.44 TestOutcome

Entity TestOutcome identifies one of a set of enumerated values to be associated with a test. The value is based on a specification of how to implement that test and the criteria for the success or failure of that test.

EXPRESS specification:

```
*)
  ENTITY TestOutcome
    SUBTYPE OF(Outcome);
      qualifier  : OPTIONAL QualifierType;
  INVERSE
    forTest      :  Test FOR allowedOutcomes;
  UNIQUE
    outcomeKey   :  allowedValue,
                    qualifier,
                    forTest;
  WHERE
    legalValues :  SELF\Outcome.allowedValue IN
```

```
                    [Pass, Fail, NotKnown, UserDefined];
  END_ENTITY;
(*
```

Attribute definitions:

qualifier : Attribute qualifier provides a method for specifying a distinct way that a test can fail that may lead to differing diagnosis. The qualifier is not simply descriptive, nor is it intended to represent degrees of confidence or measurement values/ranges for their own sake. Examples include Lo and Hi to indicate whether the test failed as a result of a measurement below the lower limit or above the higher limit, respectively.

forTest : Inverse attribute forTest links the outcome to the specific test with that outcome.

Formal propositions:

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

6.1.45 TimeCost

Entity TimeCost defines a time-related cost as a measure of the time it takes to perform a task.

EXPRESS specification:

```
*)
  ENTITY TimeCost
    SUBTYPE OF(Cost);
      timeCostUnit   : TimeUnit;
  END_ENTITY;
(*
```

Attribute definitions:

timeCostUnit : Attribute timeCostUnit defines the time units associated with the time-related cost.

6.1.46 actionDag

EXPRESS specification:

Function actionDag examines the set of Actions in a model and ensures that it is a directed acyclic graph (DAG). In other words, when traversing the child relations, it ensures that no cycles exist in the model.

```
*)
  FUNCTION actionDag
    (target      :  Action;
     descendents :  SET OF Action ) : LOGICAL;
    LOCAL
        i : INTEGER;
```

```
    END_LOCAL;
    IF ((NOT EXISTS(descendents)) OR (SIZEOF(descendents) = 0)) THEN
       RETURN (TRUE);
    END_IF;
    IF target IN descendents THEN
       RETURN (FALSE);
    END_IF;
    REPEAT i := LOINDEX(descendents) TO HIINDEX(descendents);
       IF (EXISTS(descendents[i].subAction) AND
             (actionDag(target,descendents[i].subAction)=FALSE))
       THEN
             RETURN(FALSE);
       END_IF;
    END_REPEAT;
    RETURN(TRUE);
  END_FUNCTION;
(*
```

6.1.47 actionRollup

EXPRESS specification:

Function actionRollup examines the set of Actions in a model and ensures that, when considering each of the "parts" (i.e., children) of an Action, these children are represented in the top-level set. Thus, this function ensures that the set of Actions listed with a model is a "rollup" of all of the Actions in the model.

```
*)
  FUNCTION actionRollup
       (mdl:DiagnosticModel;elem:SET [1:?] OF Action):BOOLEAN;

        LOCAL
          i : INTEGER;
          j : INTEGER;
          tmp : Action;
        END_LOCAL;

        REPEAT i := LOINDEX(elem) TO HIINDEX(elem);
          IF EXISTS(elem[i].subAction) THEN
            REPEAT j := LOINDEX(elem[i].subAction) TO
                        HIINDEX(elem[i].subAction);
              tmp := elem[i].subAction[j];
              IF NOT(tmp.partOfModel :=: mdl) THEN
                RETURN(FALSE);
              END_IF;
            END_REPEAT;
          END_IF;
        END_REPEAT;
        RETURN(TRUE);
  END_FUNCTION;
(*
```

6.1.48 cdfIntCheck

EXPRESS specification:

Function cdfIntCheck verfifes that the cdf is plausibly an integral of the pdf with respect to time. It is not possible to check that the cdf is an exact integral of the pdf. Instead, the function checks that between any two consecutive points, the cdf changes by an amount that is bounded by the maximum and minimum values of pdf at the two points.

```
*)
  FUNCTION cdfIntCheck
     (t : LIST OF TimeValue;
      pdf : LIST OF DistributionPoint;
      cdf : LIST OF ProbabilityValue):BOOLEAN;

       LOCAL
         result : BOOLEAN := TRUE;
         j : INTEGER;

         maxPdf,minPdf,dCdf,dt : REAL;
       END_LOCAL;

       (* Return false immediately if the lists are not the same length.
          *)
       IF ((SIZEOF(t)<>SIZEOF(pdf)) OR (SIZEOF(t)<>SIZEOF(cdf))) THEN
         result := FALSE;
         RETURN(result);
       END_IF;
       (* Return true immediately if the lists have size<=1 *)
       IF (SIZEOF(t)<=1) THEN
         RETURN(result);
       END_IF;

       (* Perform the detailed check at each pair of points *)
       REPEAT j := 2 TO SIZEOF(t);
         IF (pdf[j]>pdf[j-1]) THEN (* get the max and min values of the
                                      pdf at the two points*)
           minPdf:=pdf[j-1];
           maxPdf:=pdf[j];
         ELSE
           minPdf:=pdf[j];
           maxPdf:=pdf[j-1];
         END_IF;
         dCdf := cdf[j-1]-cdf[j]; (* Compute the change in the cdf *)
         dt := t[j]-t[j-1];  (* Compute the change in t *)

         (* Verify that the change in cdf is bounded by the max and min
            of pdf integrated over the change in time *)
         IF ((dCdf>dt*maxPdf) OR (dCdf<dt*minPdf)) THEN
           result := FALSE;
         END_IF;
       END_REPEAT;

       RETURN(result);
```

```
   END_FUNCTION;
(*
```

6.1.49 costLabelUnique

EXPRESS specification:

Function costLabelUnique verifies that an aggregation of cost entities has unique cost labels. It returns FALSE if two elements of c have the same costLabel attribute. It returns UNKNOWN if any element comparison is unknown. Otherwise, it returns TRUE.

```
*)
   FUNCTION costLabelUnique
      (c: AGGREGATE OF Cost) : LOGICAL;

      LOCAL
        result : LOGICAL;
        unknownp : BOOLEAN := FALSE;
        i : INTEGER;
        j : INTEGER;
      END_LOCAL;

      IF (SIZEOF(c) = 0) THEN
        RETURN(TRUE);
      END_IF;
      REPEAT i := LOINDEX(c) TO (HIINDEX(c) - i);
        REPEAT j := (i+1) TO HIINDEX(c);
          result := (c[i].costLabel :=: c[j].costLabel);
          IF (result = TRUE) THEN
            RETURN(FALSE);
          END_IF;
          IF (result = UNKNOWN) THEN
            unknownp := TRUE;
          END_IF;
        END_REPEAT;
      END_REPEAT;
      IF unknownp THEN
        RETURN(UNKNOWN);
      ELSE
        RETURN(TRUE);
      END_IF;
   END_FUNCTION;
(*
```

6.1.50 decreasingCdfCheck

EXPRESS specification:

Function decreasingcdfCheck returns false if any value in cdf is increasing. It returns true if “cdf” is monotonically nonincreasing, or SIZEOF(cdf) is 0 or 1.

```
*)
   FUNCTION decreasingCdfCheck
      (cdf : LIST OF ProbabilityValue):BOOLEAN;
```

```
      LOCAL
        result : BOOLEAN := TRUE; (* default result is true *)
        j : INTEGER;
      END_LOCAL;

      IF (SIZEOF(cdf)>1) THEN (* perform detailed check if sizeof cdf>1
                                 *)
        REPEAT j := 2 TO SIZEOF(cdf);
         IF (cdf[j] > cdf[j-1]) THEN
          result := FALSE; (* return false if any point increases *)
         END_IF;
        END_REPEAT;
      END_IF;
      RETURN(result);
  END_FUNCTION;
(*
```

6.1.51 diagnosisDag

EXPRESS specification:

Function diagnosisDag examines the set of Diagnoses in a model and ensures that it is a DAG. In other words, when traversing the child relations, it ensures that no cycles exist in the model.

```
*)
   FUNCTION diagnosisDag
     (target :  Diagnosis;
      descendents :  SET OF Diagnosis ) : LOGICAL;

      LOCAL
         i : INTEGER;
      END_LOCAL;

      IF ((NOT EXISTS(descendents)) OR (SIZEOF(descendents) = 0)) THEN
         RETURN (TRUE);
      END_IF;
      IF target IN descendents THEN
         RETURN (FALSE);
      END_IF;
      REPEAT i := LOINDEX(descendents) TO HIINDEX(descendents);
         IF (EXISTS(descendents[i].subGroup) AND
             (diagnosisDag(target,descendents[i].subGroup)=FALSE)) THEN
           RETURN(FALSE);
         END_IF;
      END_REPEAT;
      RETURN(TRUE);
   END_FUNCTION;
(*
```

6.1.52 diagnosisRollup

EXPRESS specification:

Function diagnosisRollup examines the set of Diagnoses in a model and ensures that, when considering each of the "parts" (i.e., children) of a Diagnosis, these children are represented in the top-level set. Thus, this function ensures that the set of Diagnoses listed with a model is a "rollup" of all of the Diagnoses in the model.

```
*)
  FUNCTION diagnosisRollup
        (mdl:DiagnosticModel; elem:SET [1:?] OF Diagnosis):BOOLEAN;

        LOCAL
          i : INTEGER;
          j : INTEGER;
          tmp : Diagnosis;
        END_LOCAL;

        REPEAT i := LOINDEX(elem) TO HIINDEX(elem);
          IF EXISTS(elem[i].subGroup) THEN
            REPEAT j := LOINDEX(elem[i].subGroup) TO
                        HIINDEX(elem[i].subGroup);
              tmp := elem[i].subGroup[j];
              IF NOT(tmp.partOfModel :=: mdl) THEN
                RETURN(FALSE);
              END_IF;
            END_REPEAT;
          END_IF;
        END_REPEAT;
        RETURN(TRUE);
  END_FUNCTION;
(*
```

6.1.53 increasingTimeCheck

EXPRESS specification:

Function increasingTimeCheck returns false if any value in t is non-increasing. It returns true if t is monotonically increasing, or SIZEOF(t) is 0 or 1.

```
*)
  FUNCTION increasingTimeCheck
     (t : LIST OF TimeValue):BOOLEAN;

       LOCAL
         result : BOOLEAN := TRUE;    (* default result is true *)
         j : INTEGER;
       END_LOCAL;

       IF (SIZEOF(t)>1) THEN (*perform detailed check if sizeof t>1*)
         REPEAT j := 2 TO SIZEOF(t);
           IF (t[j] <= t[j-1]) THEN
```

```
            result := FALSE;   (* return false if any point non-
                                    increasing *)
          END_IF;
        END_REPEAT;
      END_IF;
      RETURN(result);
  END_FUNCTION;
(*
```

6.1.54 itemDag

EXPRESS specification:

Function itemDag examines the set of SystemItems in a model and ensures that it is a DAG. In other words, when traversing the child relations, it ensures that no cycles exist in the model.

```
*)
  FUNCTION itemDag
     (target :  SystemItem;
      descendents :  SET OF SystemItem ) : LOGICAL;

      LOCAL
         i : INTEGER;
      END_LOCAL;

      IF ((NOT EXISTS(descendents)) OR (SIZEOF(descendents) = 0)) THEN
         RETURN (TRUE);
      END_IF;
      IF target IN descendents THEN
         RETURN (FALSE);
      END_IF;
      REPEAT i := LOINDEX(descendents) TO HIINDEX(descendents);
         IF (EXISTS(descendents[i].subAssembly) AND
             (itemDag(target,descendents[i].subAssembly)=FALSE)) THEN
            RETURN(FALSE);
         END_IF;
      END_REPEAT;
      RETURN(TRUE);
  END_FUNCTION;
(*
```

6.1.55 itemRollup

EXPRESS specification:

Function itemRollup examines the set of SystemItems in a model and ensures that, when considering each of the parts (i.e., children) of a SystemItem, these children are represented in the top-level set. Thus, this function ensures that the set of SystemItems listed with a model is a rollup of all of the SystemItems in the model.

```
*)
  FUNCTION itemRollup
        (mdl:DiagnosticModel; elem:SET [1:?] OF SystemItem):BOOLEAN;
```

```
        LOCAL
          i : INTEGER;
          j : INTEGER;
          tmp : SystemItem;
        END_LOCAL;

        REPEAT i := LOINDEX(elem) TO HIINDEX(elem);
          IF EXISTS(elem[i].subAssembly) THEN
            REPEAT j := LOINDEX(elem[i].subAssembly) TO
    HIINDEX(elem[i].subAssembly);
              tmp := elem[i].subAssembly[j];
              IF NOT(tmp.partOfModel :=: mdl) THEN
                RETURN(FALSE);
              END_IF;
            END_REPEAT;
          END_IF;
        END_REPEAT;
        RETURN(TRUE);
  END_FUNCTION;
(*
```

6.1.56 levelsAcyclic

EXPRESS specification:

Function levelsAcyclic ensures that, for a particular entity occuring at a Level, the chain of Levels does not cycle back on itself.

```
*)
  FUNCTION levelsAcyclic
       (lvl:level):BOOLEAN;
       LOCAL
         target : level := lvl;
       END_LOCAL;

       REPEAT WHILE (EXISTS(lvl.successor));
           IF (target = lvl.successor) THEN
              return(FALSE);
           END_IF;
           lvl := lvl.successor;
       END_REPEAT;
       RETURN(TRUE);
  END_FUNCTION;
END_SCHEMA;
(*
```

6.1.57 Common Element Model EXPRESS-G diagrams

The EXPRESS-G definition of the CEM is represented by Figure 4 through Figure 8.

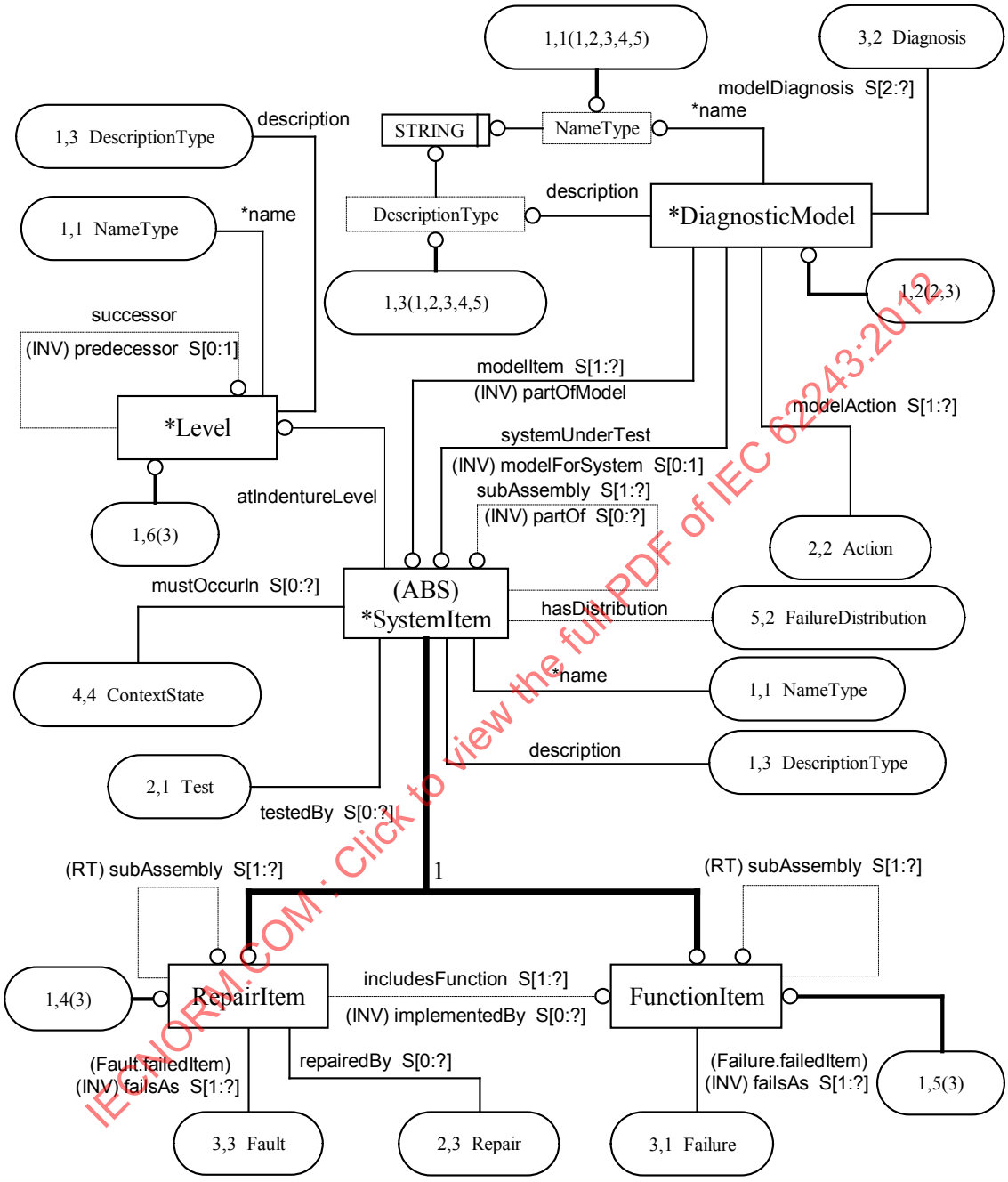


Figure 4 —AI_ESTATE_CEM EXPRESS-G diagram 1 of 5

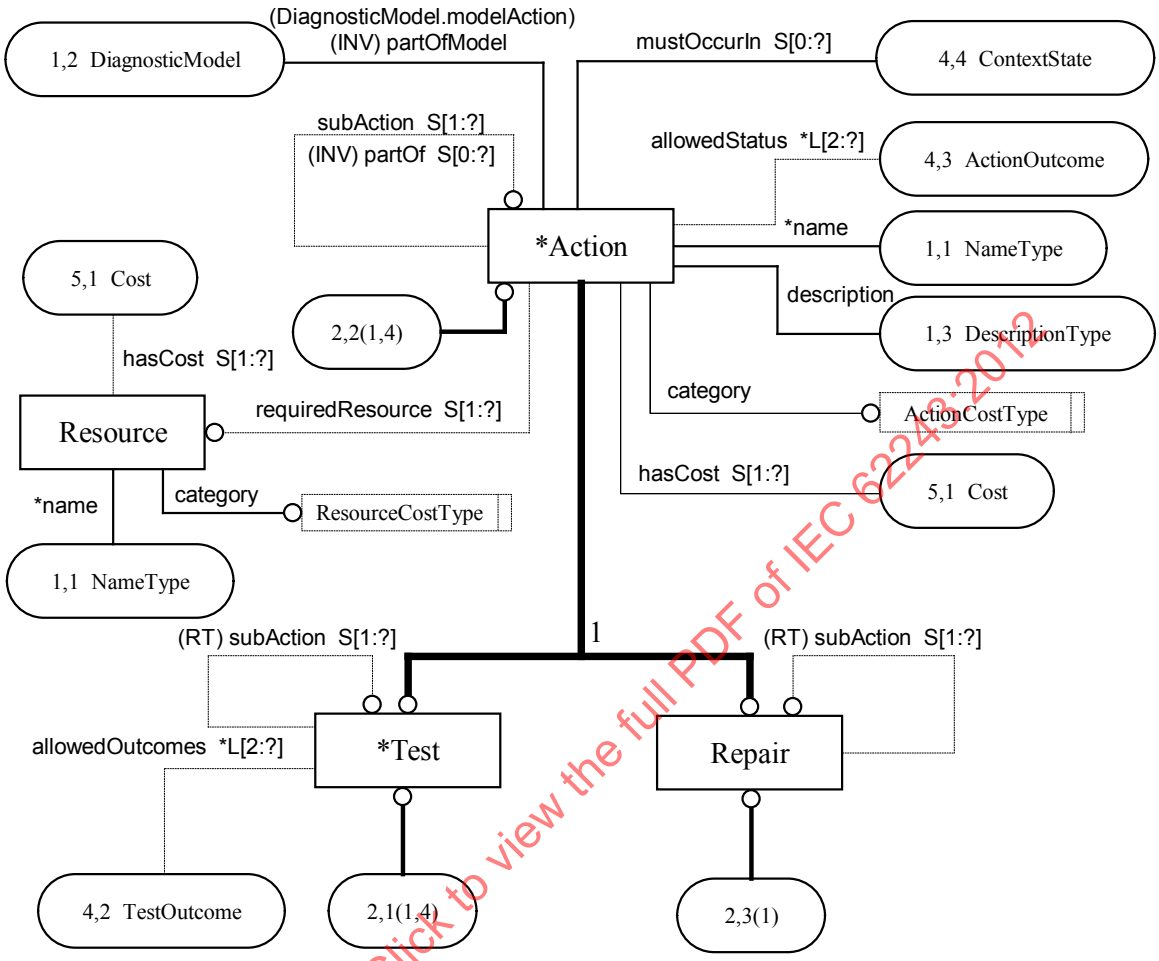


Figure 5—AI_ESTATE_CEM EXPRESS-G diagram 2 of 5

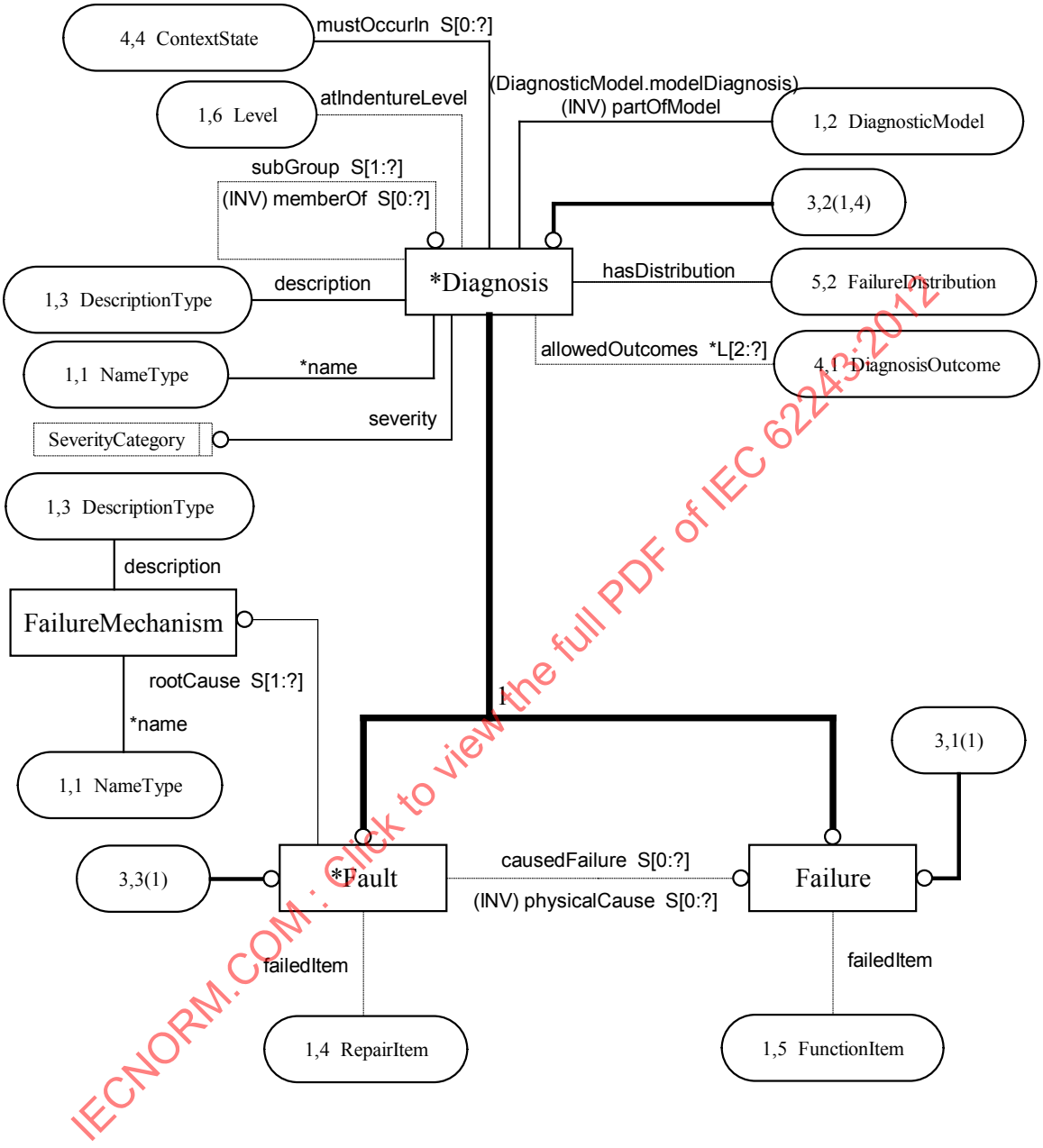


Figure 6—AI_ESTATE_CEM EXPRESS-G diagram 3 of 5

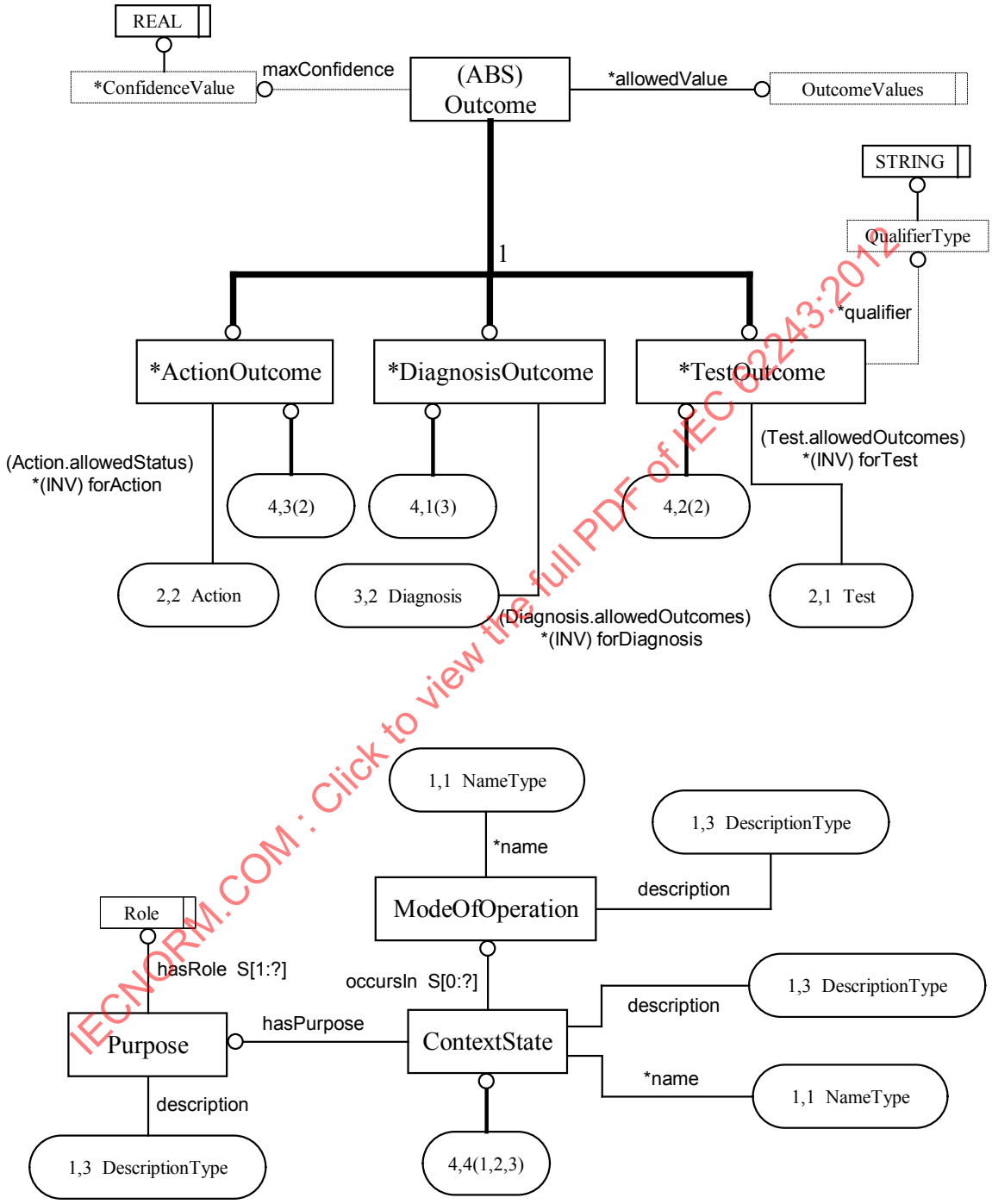


Figure 7—AI_ESTATE_CEM EXPRESS-G diagram 4 of 5

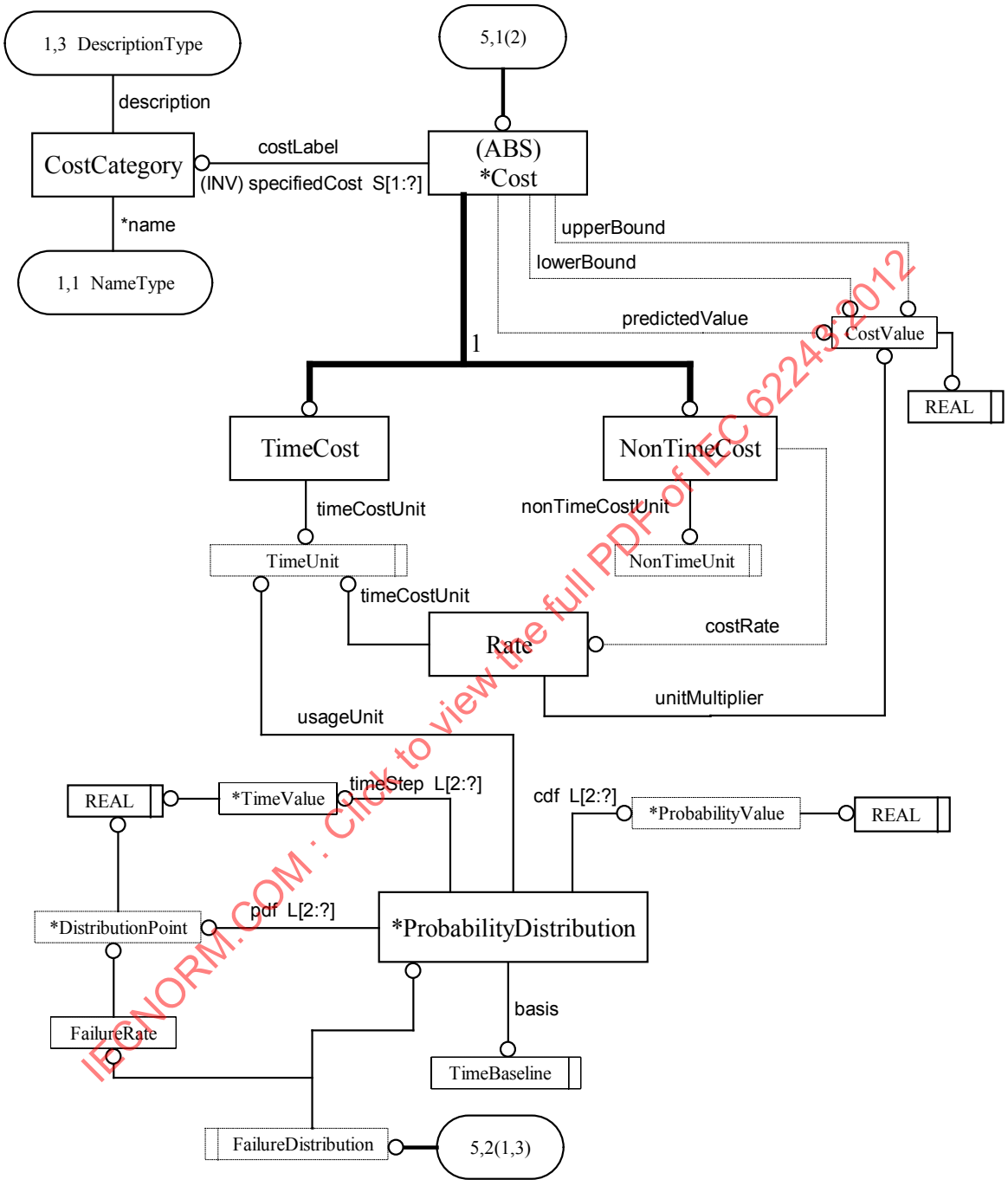


Figure 8—AI_ESTATE_CEM EXPRESS-G diagram 5 of 5

*)

(*

6.2 AI_ESTATE_BNM

Recent applications of artificial intelligence to fault diagnosis have involved using Bayesian inference to drive the diagnosis. The AI_ESTATE_BAYES_NETWORK_MODEL captures information necessary for creating diagnostic Bayesian networks. Assumptions made with this model include that random variables corresponding to diagnosis can only depend on test variables. In addition, the probability tables are to be fully explicated (including closure, i.e., summing to one across dependent joints) and array position in the probability array corresponds to array position in the dependence array. All probability distributions encoded in the network probability tables shall be multinomial distributions (i.e., distributions over a discrete set of outcomes).

EXPRESS specification:

```
*)
SCHEMA AI_ESTATE_BNM;
   REFERENCE FROM AI_ESTATE_CEM;

(*
```

6.2.1 DependentElement

Select type DependentElement corresponds to a dependent random variable in the Bayesian model. As a select type, this enables specification of a dependent random variable as a test, a diagnosis, a fault, or a failure.

EXPRESS specification:

```
*)
   TYPE DependentElement = SELECT
      (BayesTest,
      BayesDiagnosis,
      BayesFault,
      BayesFailure);
   END_TYPE;
(*
```

6.2.2 BayesDiagnosis

Entity BayesDiagnosis corresponds to an individual diagnosis within the Bayes net model. Because BayesDiagnosis is a subtype of diagnosis, it inherits all of the characteristics of a diagnosis. Note, however, that diagnostic outcomes are required in this case.

EXPRESS specification:

```
*)
   ENTITY BayesDiagnosis
      SUPERTYPE OF (ONEOF(BayesFault, BayesFailure))
      SUBTYPE OF(Diagnosis);
        SELF\Diagnosis.allowedOutcomes   : LIST [2:?] OF UNIQUE
                                           BayesDiagnosisOutcome;
   WHERE
      sumToOne  : checkDiagnosisProbabilities(allowedOutcomes);
```

```
  END_ENTITY;
(*
```

Attribute definitions:

allowedOutcomes : Attribute allowedOutcomes identifies the set of outcomes associated with this particular diagnosis. Note that the corresponding probability tables can be found on the outcome definition.

The order of outcomes in the list is significant to align the probabilities specified in BayesDiagnosisOutcome.probability. The outcomes shall correspond to the order Good, Candidate, Bad, NotKnown, and UserDefined, respectively. Skipping an outcome is permitted; however, the order must be maintained.

Formal propositions:

sumToOne Proposition sumToOne ensures that summing the probabilities in the associated probability table for a given dependent configuration results in a value of 1.0. Otherwise, the probability tables are not legal tables.

6.2.3 BayesDiagnosisOutcome

Entity BayesDiagnosisOutcome is a subtype of DiagnosisOutcome and associates legal outcomes to a BayesDiagnosis.

EXPRESS specification:

```
*)
  ENTITY BayesDiagnosisOutcome
    SUBTYPE OF(DiagnosisOutcome);
      probability   : LIST [1:?] OF ConfidenceValue;
  INVERSE
     SELF\DiagnosisOutcome.forDiagnosis  : BayesDiagnosis FOR
                                           allowedOutcomes;
  WHERE
    probabilityTable  :  SIZEOF(probability) = 1;
    noConfidence      :  NOT(EXISTS(SELF\Outcome.maxConfidence));
  END_ENTITY;
(*
```

Attribute definitions:

probability : Attribute probability associates the portion of the probability table to a random variable corresponding to this particular state of the variable. Because outcomes are unique, the list of probabilities is uniquely associated with a particular random variable, thus permitting alignment of the arrays to interpret the tables properly.

forDiagnosis : Inverse attribute forDiagnosis identifies the Bayes diagnosis to which this particular outcome is associated.

Formal propositions:

probabilityTable	Proposition probabilityTable ensures that the size of the probability table associated with this outcome is of the correct size. In particular, it ensures that a probability entry exists for every dependent configuration possible. In the case of a diagnosis outcome, it is assumed that the probabilities represent the priors in the model (i.e., diagnosis outcomes are not dependent on any other random variables in the network). Thus, it is sufficient that only one probability be stored for each outcome value.
noConfidence	Proposition noConfidence pecifies that confidence values are not used in the Bayesian model; therefore, attribute maxConfidence shall be omitted.

6.2.4 BayesFailure

Entity BayesFailure corresponds to an individual failure diagnosis within the Bayes net model. Because BayesFault is a subtype of Failure and BayesDiagnosis, it inherits all of the characteristics of BayesDiagnosis and Failure. Note, however, that diagnostic outcomes are required in this case, as required by the BayesDiagnosis supertype.

EXPRESS specification:

```
*)
  ENTITY BayesFailure
    SUBTYPE OF(BayesDiagnosis, Failure);
  WHERE
    sumToOne : checkDiagnosisProbabilities
               (SELF\BayesDiagnosis.allowedOutcomes);
  END_ENTITY;
(*
```

Formal propositions:

sumToOne	Proposition sumToOne ensures that summing the probabilities in the associated probability table for a given dependent configuration results in a value of 1.0. Otherwise, the probability tables are not legal tables.

6.2.5 BayesFault

Entity BayesFault corresponds to an individual fault diagnosis within the Bayes net model. Because BayesFault is a subtype of BayesDiagnosis and Fault, it inherits all of the characteristics of BayesDiagnosis and Fault. Note, however, that diagnostic outcomes are required in this case, as required by the BayesDiagnosis supertype.

EXPRESS specification:

```
*)
  ENTITY BayesFault
    SUBTYPE OF(BayesDiagnosis, Fault);
```

```
   WHERE
      sumToOne : checkDiagnosisProbabilities
                 (SELF\BayesDiagnosis.allowedOutcomes);
   END_ENTITY;
(*
```

Formal propositions:

sumToOne — Proposition sumToOne ensures that summing the probabilities in the associated probability table for a given dependent configuration results in a value of 1.0. Otherwise, the probability tables are not legal tables.

6.2.6 BayesNetworkModel

Entity BayesNetworkModel is the starting point of the Bayes network. It provides the rollup of the random variables (i.e., tests and diagnosis) used in diagnosis. Because this is a subtype of diagnostic_model, the BayesNetworkModel inherits all other characteristics of a diagnostic model from the AI_ESTATE_CEM.

EXPRESS specification:

```
*)
   ENTITY BayesNetworkModel
      SUBTYPE OF(DiagnosticModel);
      SELF\DiagnosticModel.modelAction     : SET [2:?] OF BayesTest;
      SELF\DiagnosticModel.modelDiagnosis  : SET [2:?] OF BayesDiagnosis;
   END_ENTITY;
(*
```

Attribute definitions:

modelAction : Attribute modelAction identifies the set of Bayes tests included in the diagnostic model. Because the Bayes network focuses on random variables, it is assumed that these tests are to be treated as leaves in the test hierarchy.

modelDiagnosis : Attribute modelDiagnosis identifies the set of Bayes diagnosis included in the diagnostic model. Because the Bayes network focuses on random variables, it is assumed that these tests are to be treated as leaves in the diagnosis hierarchy.

6.2.7 BayesTest

Entity BayesTest corresponds to an individual test within the Bayes net model. Because Bayes_test is a subtype of test, it inherits all of the characteristics of a diagnostic test. Note, however, that test outcomes are required in this case.

EXPRESS specification:

```
*)
   ENTITY BayesTest
      SUBTYPE OF(Test);
      dependsOnElement           : LIST OF UNIQUE DependentElement;
      SELF\Test.allowedOutcomes  : LIST [2:?] OF UNIQUE BayesTestOutcome;
   WHERE
      sumToOne  :  checkTestProbabilities(allowedOutcomes);
```

```
  END_ENTITY;
(*
```

Attribute definitions:

dependsOnElement : Attribute dependsOnElement identifies the set of elements (either tests or diagnoses) upon which this particular test depends. Note that dependence, in this context, refers to conditional dependence as defined for Bayesian networks.

allowedOutcomes : Attribute allowedOutcomes identifies the set of outcomes associated with this particular test. Note that the corresponding probability tables can be found on the outcome definition.

The order of outcomes in the list is significant to align the probabilities specified in BayesTestOutcome.probability. The outcomes shall correspond to the order Pass, Fail, NotKnown, and UserDefined, respectively. Skipping an outcome is permitted; however, the order must be maintained.

Formal propositions:

sumToOne Proposition sumToOne ensures that summing the probabilities in the associated probability table for a given dependent configuration results in a value of 1.0. Otherwise, the probability tables are not legal tables.

6.2.8 BayesTestOutcome

Entity BayesTestOutcome is a subtype of "TestOutcome" and associates legal outcomes to a BayesTest.

EXPRESS specification:

```
*)
  ENTITY BayesTestOutcome
    SUBTYPE OF(TestOutcome);
      probability : LIST [1:?] OF ConfidenceValue;
  INVERSE
    SELF\TestOutcome.forTest   : BayesTest FOR allowedOutcomes;
  WHERE
    probabilityTable  : SIZEOF(probability) =
                        variableSize(SELF\TestOutcome.forTest.dependsOn
                        Element);
    noConfidence      : NOT(EXISTS(SELF\Outcome.maxConfidence)) ;
  END_ENTITY;
(*
```

Attribute definitions:

probability : Attribute probability associates the portion of the probability table to a random variable corresponding to this particular state of the variable. Because outcomes are unique, the list of probabilities is uniquely associated with a particular random variable, thus permitting alignment of the arrays to properly interpret the tables. To ensure proper ordering, the list of values follows the order specified by the dependsOnElement attribute with associated outcomes corresponding to the order,

Good, Candidate, Bad, NotKnown, and UserDefined for diagnoses and Pass, Fail, NotKnown, and UserDefined for tests. Outcomes may be skipped, but the order must be maintained.

forTest : Inverse attribute forTest identifies the BayesTest to which this particular outcome is associated.

Formal propositions:

probabilityTable — Proposition probabilityTable ensures that the size of the probability table associated with this outcome is of the correct size. In particular, it ensures that a probability entry exists for every dependent configuration possible.

For test outcomes, each outcome is conditioned on the combination of parent random variables (test outcomes or diagnosis outcomes). Therefore, the number of conditional probabilities for each outcome is the product of the number of outcomes for each of the parents.

noConfidence — Proposition noConfidence pecifies that confidence values are not used in the Bayesian model; therefore, attribute maxConfidence shall be omitted.

6.2.9 ModelRule

Rule ModelRule shall apply to the population of DiagnosticModel entities in a BNM exchange file.

EXPRESS specification:

```
*)
   RULE ModelRule FOR
      (DiagnosticModel);
   WHERE
      oneModel  :  SIZEOF(DiagnosticModel) = 1;
      onlySubtype  : SIZEOF(QUERY(tmp <* DiagnosticModel |
                     NOT('AI_ESTATE_BNM.BAYESNETWORKMODEL' IN
                     TYPEOF(tmp)))) = 0;
   END_RULE;
(*
```

6.2.10 checkDiagnosisProbabilities

Function checkDiagnosisProbabilities takes a dependent configuration for an entry in the conditional probability table and ensures that the probabilities over that configuration sum to one. If they do not sum to one, the corresponding probability table is not a legal table.

EXPRESS specification:

```
*)
   FUNCTION checkDiagnosisProbabilities
        (out : LIST [2:?] OF BayesDiagnosisOutcome) : BOOLEAN;
        LOCAL
          legal : BOOLEAN := TRUE;
          sum : LIST [1:?] OF REAL;
        END_LOCAL;
```

```
      REPEAT y := LOINDEX(out[1].probability) TO
                  HIINDEX(out[1].probability);
        sum[y] := 0;
      END_REPEAT;
      REPEAT x := LOINDEX(out) TO HIINDEX(out);
        REPEAT y := LOINDEX(out[x].probability) TO
    HIINDEX(out[x].probability);
          sum[y] := sum[y] + out[x].probability[y];
        END_REPEAT;
      END_REPEAT;
      REPEAT y := LOINDEX(out[1].probability) TO
                  HIINDEX(out[1].probability);
        IF (sum[y] <> 1.0) THEN
            legal := FALSE;
        END_IF;
      END_REPEAT;
      RETURN (legal);

  END_FUNCTION;
(*
```

6.2.11 checkTestProbabilities

Function checkTestProbabilities takes a dependent configuration for an entry in the conditional probability table and ensures that the probabilities over that configuration sum to one. If they do not sum to one, the corresponding probability table is not a legal table.

EXPRESS specification:

```
*)
  FUNCTION checkTestProbabilities
      (out : LIST [2:?] OF BayesTestOutcome) : BOOLEAN;
      LOCAL
        legal : BOOLEAN := TRUE;
        sum : LIST [1:?] OF REAL;
      END_LOCAL;

      REPEAT y := LOINDEX(out[1].probability) TO
                  HIINDEX(out[1].probability);
        sum[y] := 0;
      END_REPEAT;
      REPEAT x := LOINDEX(out) TO HIINDEX(out);
        REPEAT y := LOINDEX(out[x].probability) TO
                    HIINDEX(out[x].probability);
          sum[y] := sum[y] + out[x].probability[y];
        END_REPEAT;
      END_REPEAT;
      REPEAT y := LOINDEX(out[1].probability) TO
                  HIINDEX(out[1].probability);
        IF (sum[y] <> 1.0) THEN
            legal := FALSE;
        END_IF;
      END_REPEAT;
```

```
        RETURN (legal);

   END_FUNCTION;
(*
```

6.2.12 variableSize

Function variableSize computes the product of the sizes of the dependent variable configurations. In other words, it multiplies the number of outcomes associated with each dependent variable together to determine how large the corresponding probability table must be.

EXPRESS specification:

```
*)
   FUNCTION variableSize
        (elem : LIST [1:?] OF dependentElement) : NUMBER;
        LOCAL
          size : NUMBER := 1;
        END_LOCAL;

        IF (SIZEOF(elem) > 0) THEN
          REPEAT x := LOINDEX(elem) TO HIINDEX(elem);
            IF ('AI_ESTATE_BNM.BAYESTEST' IN typeof(elem[x])) THEN
              size := size * SIZEOF(elem[x]\bayesTest.allowedOutcomes);
            ELSE
              size := size *
                      SIZEOF(elem[x]\bayesDiagnosis.allowedOutcomes);
            END_IF;
          END_REPEAT;
        END_IF;
        RETURN(size);
   END_FUNCTION;
END_SCHEMA;
(*
```

6.2.13 Bayesian Network Model EXPRESS-G diagrams

The EXPRESS-G definition of the BNM is represented by Figure 9.

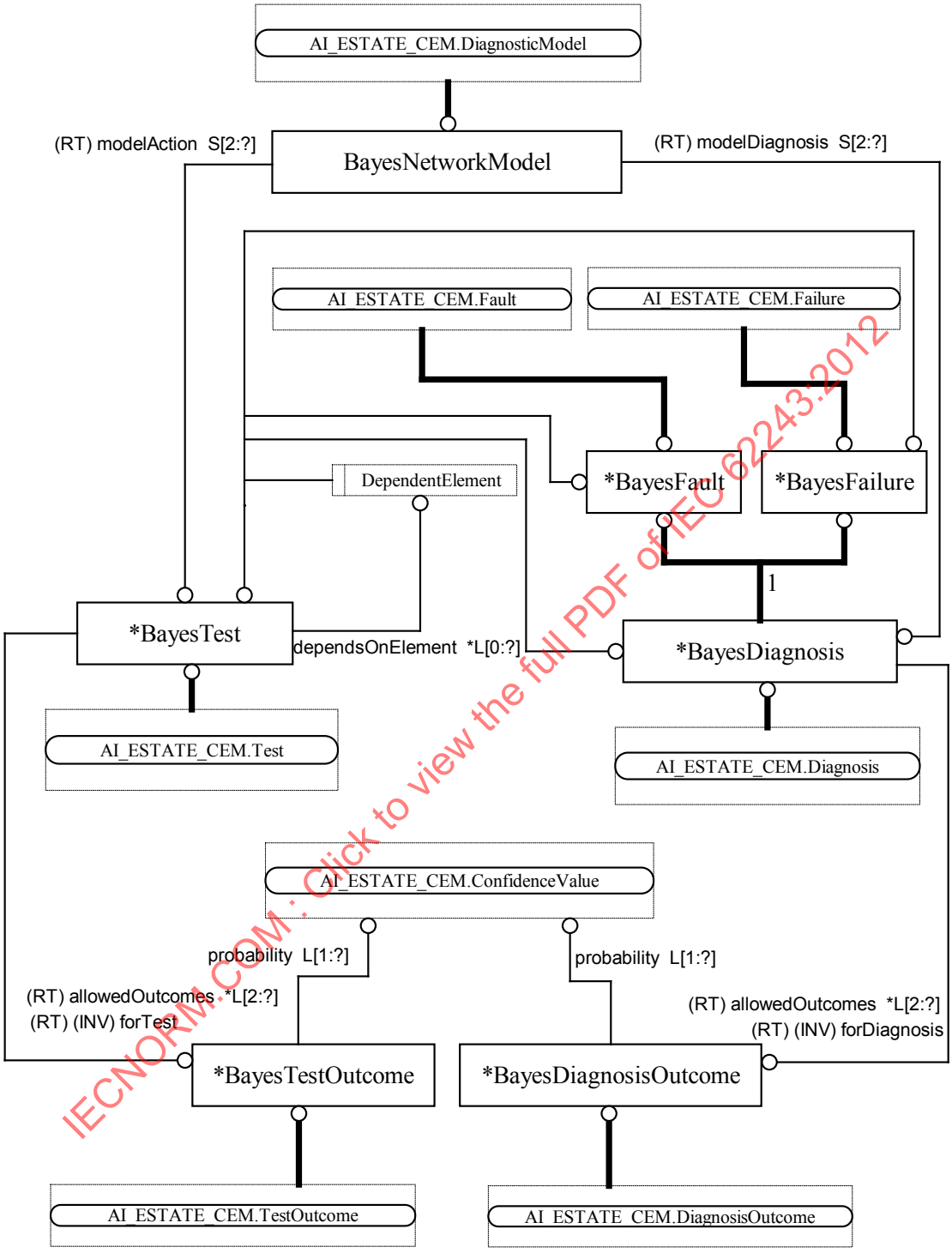


Figure 9—AI_ESTATE_BNM EXPRESS-G diagram 1 of 1

*)

(*

6.3 AI_ESTATE_DIM

The AI-ESTATE D-Matrix Inference Model information model is defined in 6.3.1 through 6.3.6. The expectation is that this model represents a "canonical" D (i.e., dependency) matrix from the perspective of what can be inferred from test outcomes. Specifically, in a canonical D-matrix, test outcomes are limited to Pass or Fail outcomes and diagnosis outcomes are limited to Good and Candidate outcomes. The columns of the matrix correspond to tests and rows can be either tests or diagnoses. It is required that, if a test passes, then all of the "column" inferences drawn in the matrix for that test be logically ANDed together, and all of the "row" inferences drawn for that test must be "ORed" together. Conversely, if a test fails, it is required that all of the "column" inferences drawn in the matrix for that test be "ORed" together, and all of the "row" inferences for that test must be "ANDed" together. It is also required that inferences be of like type (i.e., Pass => Pass/Good or Fail => Fail/Candidate).

The set of inferences associated with a particular test outcome are limited either to a list of conjuncted inferences or a list of disjuncted inferences (not both). Determining conjuncted versus disjuncted depends on the outcome of the test. Specifically, only inferences from passing tests are conjuncted and can only lead to pass/good outcomes on tests and diagnoses, respectively. In addition, inference from failing tests are disjuncted and can only lead to fail/candidate outcomes on tests or diagnoses, respectively.

All test inferences are required to be symmetric. In other words, the set of outcomes arising from a test passing complement the inferences arising from a test failing.

EXPRESS specification:

```
*)
SCHEMA AI_ESTATE_DIM;

   REFERENCE FROM AI_ESTATE_CEM;

(*
```

6.3.1 CellOutcome

Select type CellOutcome identifies a specific outcome that is being inferred as a result of a test outcome. As a select type, the value corresponds to either a test outcome or a diagnosis outcome.

EXPRESS specification:

```
*)
   TYPE CellOutcome = SELECT
      (DiagnosisOutcome,
      TestOutcome);
   END_TYPE;
(*
```

6.3.2 DmatrixInferenceModel

Entity DmatrixInferenceModel represents the constituents of the DIM. The model shall represent a "canonical" D (i.e., dependency) matrix from the perspective of what can be inferred from test outcomes. Specifically, in a canonical D-matrix, the columns correspond to tests. Rows of the matrix can be either tests or diagnoses. It is required that, if a test passes, then all of the "column" inferences drawn in the matrix be logically ANDed together. Conversely, if a test fails, it is required that all of the "column" inferences drawn in the matrix be "ORed" together. It is also required that inferences be of like type (i.e., Pass => Pass/Good or Fail => Fail/Candidate).

EXPRESS specification:

```
*)
   ENTITY DmatrixInferenceModel
     SUBTYPE OF(DiagnosticModel);
       testColumn   : SET [2:?] OF OutcomeInference;
   END_ENTITY;
(*
```

Attribute definitions:

testColumn : Attribute testColumn identifies the set of outcome_inference that comprise the model of the system under test. To be useful, a model shall consist of at least two inferences, corresponding to inferences from the minimum number of outcomes for the minimum number of tests in the model.

6.3.3 Inference

Entity Infrence identifies an Outcome to be inferred as a result of some other Outcome in the model being asserted. Only TestOutcomes or DiagnosisOutcomes can be inferred.

EXPRESS specification:

```
*)
   ENTITY Inference;
     cell        : CellOutcome;
   INVERSE
     assertion   : OutcomeInference FOR andOrRows;
   WHERE
     consistentOutcome   : ((SELF.assertion.preconditionTestOutcome
                           .allowedValue = Pass) AND
                            (SELF.cell\Outcome.allowedValue = Pass)) OR
                           ((SELF.assertion.preconditionTestOutcome
                           .allowedValue = Pass) AND
                            (SELF.cell\Outcome.allowedValue = Good)) OR
                           ((SELF.assertion.preconditionTestOutcome
                           .allowedValue = Fail) AND
                            (SELF.cell\Outcome.allowedValue = Fail)) OR
                           ((SELF.assertion.preconditionTestOutcome
                           .allowedValue = Fail) AND
                            (SELF.cell\Outcome.allowedValue =
                           Candidate));
   END_ENTITY;
(*
```

Attribute definitions:

cell : Attribute cell identifies a specific inference in the current logical expression based on its location in the Dmatrix.

assertion : Attribute assertion identifies the top level of the logical expression being inferred from some test outcome.

Formal propositions:

consistentOutcome	Proposition consistentOutcome specifies that the outcomes be of like type. In other words, Pass implies Pass and Fail implies Fail.

6.3.4 OutcomeInference

Entity OutcomeInference identifies an outcome to be inferred as a result of some other outcome in the model being asserted. Only TestOutcomes and DiagnosisOutcomes can be inferrred.

EXPRESS specification:

```
*)
   ENTITY OutcomeInference;
      andOrRows                  : SET [1:?] OF Inference;
      preconditionTestOutcome    : TestOutcome;
      confidence                 : OPTIONAL ConfidenceValue;
      andOrRelation              : BOOLEAN;
   UNIQUE
      oneOutcome                 : preconditionTestOutcome;
   WHERE
      conjunctOrDisjunct  : ((SELF.preconditionTestOutcome.allowedValue =
                            Pass) AND
                             (SELF.andOrRelation = TRUE)) XOR
                            ((SELF.preconditionTestOutcome.allowedValue =
                            Fail) AND
                             (SELF.andOrRelation = FALSE));
      noUserDefined       : preconditionTestOutcome.allowedValue <>
                            UserDefined;
   END_ENTITY;
(*
```

Attribute definitions:

andOrRows	:	Attribute andOrRows identifies a set of simple terms that are either ANDed or ORed together, depending on the value of the andOrRelation attribute. If andOrRelation = TRUE, then the set of terms identified here is ANDed. If andOrRelation = FALSE, then the set of terms identified here is ORed.
preconditionTestOutcome	:	Attribute preconditionTestOutcome identifies a particular outcome of the value of the TestOutcome.forTest attribute to which the value of this attribute applies, where the TestOutcome.forTest attribute identifies a particular test.
confidence	:	Attribute confidence identifies the statistical confidence in the logical expression from the outcome in outcome_inference.
andOrRelation	:	Attribute andOrRelation determines if SELF.conjunctDisjunct is ANDed or ORed. If andOrRelation = TRUE, then SELF.conjunctDisjunct is the conjunction of the members of the set. If andOrRelation = FALSE, then SELF.conjunctDisjunct is the disjunction of the members of the set.

Formal propositions:

conjunctOrDisjunct	Proposition conjunctOrDisjunct constrains the outcome inference list to be a set of conjuncts or a set of disjuncts, but not both based on whether the preconditionTestOutcome is Pass or Fail. If it is Pass, then the list must be a conjunct list (i.e., andOrRelation = TRUE). If it is Fail, then the list must be a disjunct list (i.e., andOrRelation = FALSE).
noUserDefined	Proposition noUserDefined requires that the test outcome be one of the basic standard values as defined in the Common Element Model.

6.3.5 ModelRule

Rule ModelRule shall apply to the population of DiagnosticModel entities in a DIM exchange file.

EXPRESS specification:

```
*)
   RULE ModelRule FOR
      (DiagnosticModel);
   WHERE
      oneModel  :  SIZEOF(DiagnosticModel) = 1;
      onlySubtype  : SIZEOF(QUERY(tmp <* DiagnosticModel |
                     NOT('AI_ESTATE_DIM.DMATRIXINFERENCEMODEL' IN
                     TYPEOF(tmp)))) = 0;
      END_RULE;
END_SCHEMA;
(*
```

6.3.6 D-Matrix Inference Model EXPRESS-G diagrams

The EXPRESS-G definition of the DIM is represented by Figure 10.

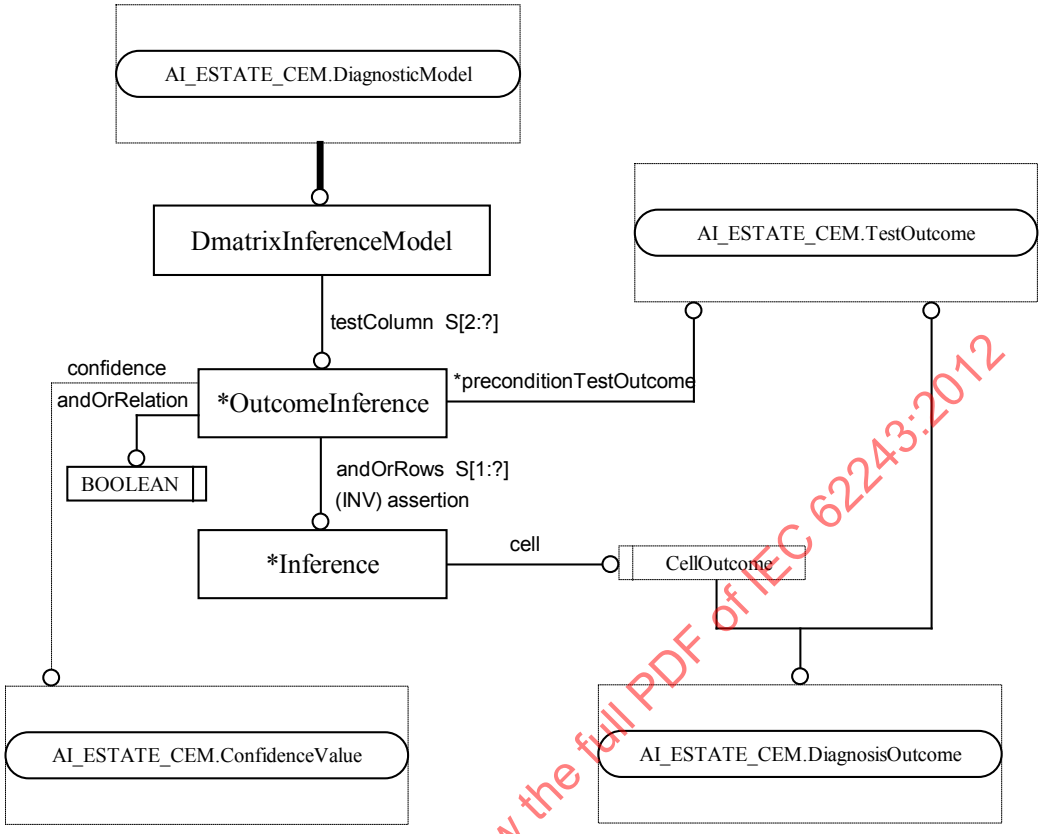


Figure 10—AI_ESTATE_DIM EXPRESS-G diagram 1 of 1

```
*)

(*
```

6.4 AI_ESTATE_DLM

The AI-ESTATE Diagnostic Logic Model information model is defined below. The constructs of this model were derived with the intent of providing a means for representing arbitrary logical expressions. The model utilizes many of the constructs defined in the AI-ESTATE Common Element Model.

An inference is drawn using a logical relationship between two or more outcomes. The set of possible inferences associated with a particular outcome are represented as expressions of the form Outcome => AND(OR(outcomes)) or Outcome => OR(AND(outcomes)).

EXPRESS specification:

```
*)
SCHEMA AI_ESTATE_DLM;
   REFERENCE FROM AI_ESTATE_CEM;

(*
```

6.4.1 DiagnosticLogicModel

Entity DiagnosticLogicModel represents the constituents of a generic Diagnostic Logic Model.

EXPRESS specification:

```
*)
   ENTITY DiagnosticLogicModel
     SUBTYPE OF(DiagnosticModel);
       inferences : SET [2:?] OF OutcomeInference;
   END_ENTITY;
(*
```

Attribute definitions:

inferences : Attribute inferences identify the set of outcome_inference that comprise the model of the system under test. To be useful, a model shall consist of at least two inferences, corresponding to inferences from the minimum number of outcomes for the minimum number of tests in the model.

6.4.2 Inference

Entity Inference identifies a set of outcome inferences to be drawn as a result of some other outcome in the model being asserted. The set of outcomes is either ANDed or ORed, as specified by the andOrRelation attribute. Only test outcomes or diagnosis outcomes can be inferred.

EXPRESS specification:

```
*)
   ENTITY Inference;
     term            : SET [1:?] OF Outcome;
     andOrRelation   : BOOLEAN;
   INVERSE
     assertion       : OutcomeInference FOR andOrTerms;
   WHERE
     noActionOutcome : SIZEOF(QUERY(tmp <* SELF.term |
                       'AI_ESTATE_DLM.ACTIONOUTCOME' IN TYPEOF(tmp))) =
                       0;
   END_ENTITY;
(*
```

Attribute definitions:

term : Attribute term identifies a set of specific outcomes to be inferred in the current logical expression.

andOrRelation : Attribute andOrRelation determines if SELF.term is ANDed or ORed. If andOrRelation = TRUE, then SELF.term is the conjunction of the members of the set. If andOrRelation = FALSE, then SELF.term is the disjunction of the members of the set.

assertion : Inverse attribute assertion identifies the top level of the logical expression being inferred from some outcome.

Formal propositions:

noActionOutcome Proposition noActionOutcome ensures that there are no action outcomes used in inference within this model.

6.4.3 OutcomeInference

Entity OutcomeInference pairs a particular outcome of a particular test with a set of inferences represented in a conjunct/disjunct form. Each inference entity of the set is a set of outcomes that are ANDed or ORed together. The OutcomeInference entity then ANDs or ORs the set of inferences.

EXPRESS specification:

```
*)
   ENTITY OutcomeInference;
      andOrTerms            : SET [1:?] OF Inference;
      preconditionOutcome   : Outcome;
      confidence            : OPTIONAL confidenceValue;
      andOrRelation         : BOOLEAN;
   UNIQUE
      one_outcome           : preconditionOutcome;
   END_ENTITY;
(*
```

Attribute definitions:

andOrTerms : Attribute andOrTerms identifies a set of composite terms that are either ANDed or ORed together, depending on the value of the andOrTerms attribute.

preconditionOutcome : Attribute preconditionOutcome identifies a particular outcome of an action, test, repair (which is a type of action), or diagnosis to which the value of this attribute applies, where the xxxOutcome.forXxx attribute (where xxx = action, test, or diagnosis) identifies a particular action, test, or diagnosis.

confidence : Attribute confidence identifies the statistical confidence in the logical expression from the outcome in outcome_inference.

andOrRelation : Attribute andOrRelation determines if SELF.conjunctDisjunct is ANDed or ORed. If andOrRelation = TRUE, then SELF.conjunctDisjunct is the conjunction of the members of the set. If andOrRelation = FALSE, then SELF.conjunctDisjunct is the disjunction of the members of the set.

6.4.4 ModelRule

Rule ModelRule shall apply to the population of DiagnosticModel entities in a DLM exchange file.

EXPRESS specification:

```
*)
   RULE ModelRule FOR
      (DiagnosticModel);
   WHERE
      oneModel  :  SIZEOF(DiagnosticModel) = 1;
      onlySubtype  : SIZEOF(QUERY(tmp <* DiagnosticModel |
                     NOT('AI_ESTATE_DLM.DIAGNOSTICLOGICMODEL' IN
                     TYPEOF(tmp)))) = 0;
   END_RULE;
END_SCHEMA;
(*
```

6.4.5 Diagnostic Logic Model EXPRESS-G diagrams

The EXPRESS-G definition of the DLM is represented by Figure 11.

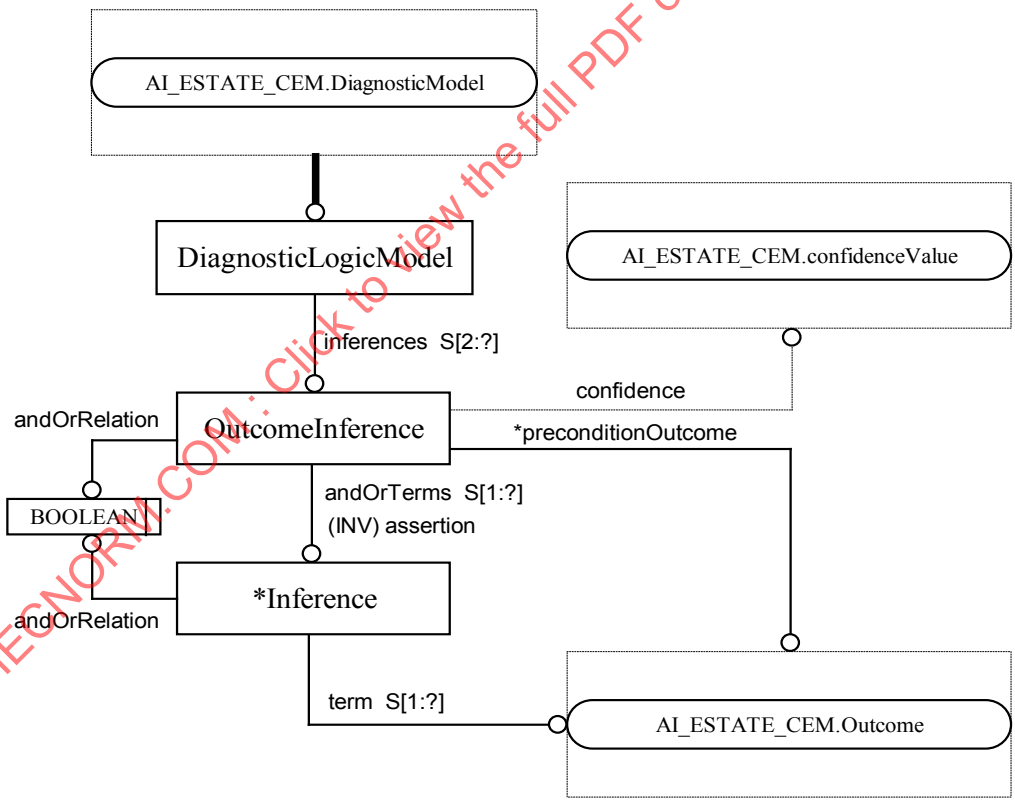


Figure 11 —AI_ESTATE_DLM EXPRESS-G diagram 1 of 1

```
*)

(*
```

6.5 AI_ESTATE_FTM

This subclause defines the AI-ESTATE Fault Tree Model information model. The constructs defined here are specific to the Fault Tree approach to system test and diagnosis. In this diagnostic method, a decision tree with fixed fault isolation strategies is constructed a priori and remains static during the diagnosis. The Fault Tree provides a test strategy that can be used without the aid of a reasoning system. This specification is included in the AI-ESTATE standard because it is frequently used as the primary diagnostic strategy or in conjunction with other test generation strategies.

The structure of a fault tree can be viewed as a decision tree. The interior nodes of the tree correspond to the different tests to be run during the fault isolation procedure. Each branch from a node corresponds to one of the possible outcomes for that test. The branch is taken that corresponds to the outcome that occurs as the result of executing the test. TestResult identifies the next test node in the tree to which to proceed, identifies the diagnostic conclusion that is drawn, or simply provides information on the status of fault isolation.

The Fault Tree Model draws on elements of the AI_ESTATE_CEM. The Test entity corresponds to a node or step such as that described previously. Each branch points to the TestResult entity in the model. A reasoner utilizes a fault tree by recommending an entry point (action) to the client application. Upon acknowledgment from the client, the fault tree is processed by starting at that entry point, executing the test associated with that step and proceeding with the actions prescribed for the outcome that results. When another step of the fault tree appears for the resulting test outcome, execution of the fault tree proceeds to that fault tree step. Eventually, the result for the test outcome should identify the diagnosis; at this point (when no other fault tree steps appear), processing of the fault tree is complete. The current diagnosis should be included at all steps of the fault tree.

EXPRESS specification:

```
*)
SCHEMA AI_ESTATE_FTM;

   REFERENCE FROM AI_ESTATE_CEM;

(*
```

6.5.1 FaultTreeModel

Entity FaultTreeModel represents the fault tree at the highest level of abstraction. Thus, it defines an entry point into the fault tree by identifying the first step (i.e., the root) of the tree. Multiple entry points can be defined, but to maintain acceptable form, they should be treated as separate models.

EXPRESS specification:

```
*)
   ENTITY FaultTreeModel
      SUBTYPE OF(DiagnosticModel);
        entryPoints  : LIST [1:?] OF UNIQUE StartingPoint;
   END_ENTITY;
(*
```

Attribute definitions:

entryPoints : Attribute entryPoints defines one or more starting points for a fault tree. As a list, entry points can be specified in sequential order based on implementation-specific needs. Should ordering not be required, the list can still be scanned to find the appropriate entry point.

6.5.2 FaultTreeStep

Entity FaultTreeStep represents a decision node in the fault tree table. Its entries identify the test to be run at this step in the fault tree and which test result/action pairs to follow. The testStep attribute uses the test entity of the AI-ESTATE Common Element Model.

EXPRESS specification:

```
*)
   ENTITY FaultTreeStep;
      result          : SET [2:?] OF TestResult;
      testStep        : Test;
      preActions      : LIST OF Action;
      postActions     : LIST OF Action;
   INVERSE
      startedBy       : SET [0:1] OF StartingPoint FOR firstStep;
      previousResult  : SET [0:1] OF TestResult FOR nextStep;
   WHERE
      outcomesAreValid  : EXISTS(testStep.allowedOutcomes) AND
                          (resultOutcomes(result) =
                          testStep.allowedOutcomes);
      oneInverse        : (SIZEOF(startedBy) = 1) OR
                          (SIZEOF(previousResult) = 1);
   END_ENTITY;
(*
```

Attribute definitions:

result : Attribute result is a set of at least two TestResult entities that comprise the outcome/action pairs for the test identified in the testStep attribute. There should be a TestResult entity in the result set for each possible outcome of the test in testStep.

testStep : Attribute testStep identifies the test that is to be run for this step of the fault tree.

preActions : Attribute preActions provides a list of actions that must be performed immediately prior to the Test specified by testStep.

postActions : Attribute postActions provides a list of actions that must be performed immediately after completing the Test specified by testStep.

startedBy : Inverse attribute startBy identifies the action corresponding to the entry point to the fault tree. If the FaultTreeStep is not an entry point, this attribute is empty.

previousResult : Attribute previousResult identifies the result in the fault tree that leads to the new step in the fault tree.

Formal propositions:

outcomesAreValid	Proposition outcomesAreValid verifies that there exists a legal outcome for the test for every TestResult specified at this step in the fault tree. The rule is satisfied when the set of outcomes equals the set returned by the function resultOutcomes.
oneInverse	Proposition oneInverse ensures that at least one of the two inverse attributes is instantiated.

6.5.3 StartingPoint

Entity StartingPoint is a subtype of Action corresponding to an entry point for fault tree-based diagnosis. As an Action, a diagnostic reasoner can recommend one or more entry points, and the entry point taken can be recorded in a Session of the DCM.

EXPRESS specification:

```
*)
  ENTITY StartingPoint
    SUBTYPE OF(Action);
     firstStep  :  FaultTreeStep;
  WHERE
    actionOnly  : NOT('AI_ESTATE_CEM.TEST' IN typeof(SELF)) AND
                  NOT('AI_ESTATE_CEM.REPAIR' IN typeof(SELF));
  END_ENTITY;
(*
```

Attribute definitions:

firstStep : Attribute firstStep points to the root of the fault tree or subtree.

Formal propositions:

actionOnly	Proposition actionOnly prevents a StartingPoint from being instantiated as either a Repair or a Test.

6.5.4 TestResult

Entity TestResult provides the outcome associated with a test. That outcome is then paired with the corresponding test result, indicating the next step in the tree by pointing to that step. If appropriate, the next step of the fault tree to which execution should proceed is identified in the nextStep attribute.

EXPRESS specification:

```
*)
  ENTITY TestResult;
    nextStep                : OPTIONAL FaultTreeStep;
    testOut                 : TestOutcome;
```

```
    currentDiagnosisOutcome  : SET OF DiagnosisOutcome;
  INVERSE
    associatedStep           : FaultTreeStep FOR result;
  WHERE
    leavesHaveDiagnoses        :    (EXISTS(nextStep)) OR
                               (SIZEOF(currentDiagnosisOutcome) > 0);
  END_ENTITY;
(*
```

Attribute definitions:

nextStep	:	Attribute nextStep identifies which FaultTreeStep to execute next when the outcome in TestOutcome results from the execution of the test of this FaultTreeStep. This attribute is optional. When no FaultTreeStep is identified, the TestResult entity is a leaf of the fault tree.
testOut	:	Attribute testOut identifies the outcome of the test of this FaultTreeStep to which this construct applies.
currentDiagnosisOutcome	:	Attribute currentDiagnosisOutcome identifies the diagnosis elements in the model that are indicted (i.e., accused) by the sequence of tests leading up to this point in the fault tree. This attribute is used to report the diagnosis resulting from traversing the tree.
associatedStep	:	Attribute associatedStep identifies the step with which the current result is associated. Because this is not a set, it enforces the tree structure of the fault tree (i.e., it is not a decision graph).

Formal propositions:

leavesHaveDiagnoses	Proposition leavesHaveDiagnoses constrains the currentDiagnosisOutcomes attribute that is a required attribute such that the associated list can be empty if associated with an internal node of the tree, but if the node is a leaf (i.e., a terminal step in the tree), then the currentDiagnosisOutcomes list cannot be empty.

6.5.5 ModelRules

Rule ModelRule shall apply to the population of DiagnosticModel entities in a FTM exchange file.

EXPRESS specification:

```
*)
  RULE ModelRules FOR
    (DiagnosticModel);
  WHERE
    oneModel  :  SIZEOF(DiagnosticModel) = 1;
    onlySubtype  : SIZEOF(QUERY(tmp <* DiagnosticModel |
                   NOT('AI_ESTATE_FTM.FAULTTREEMODEL' IN TYPEOF(tmp))))
                   = 0;
  END_RULE;
(*
```

6.5.6 resultOutcomes

EXPRESS specification:

Function resultOutcomes takes a set of test results and returns the corresponding set of test outcomes to ensure that the outcomes listed correspond to the outcomes available at the step in the tree.

```
*)
   FUNCTION resultOutcomes
        (results:SET [0:?] OF testResult) : LIST [0:?] OF testOutcome;
        LOCAL
          tOut: LIST [0:?] OF testOutcome := [];
        END_LOCAL;

        REPEAT i := LOINDEX(results) TO HIINDEX(results);
          tOut := tOut + results[i].testOut;
        END_REPEAT;
        RETURN(tOut);
   END_FUNCTION;
END_SCHEMA;
(*
```

6.5.7 Fault Tree Model EXPRESS-G diagrams

The EXPRESS-G definition of the FTM is represented by Figure 12.

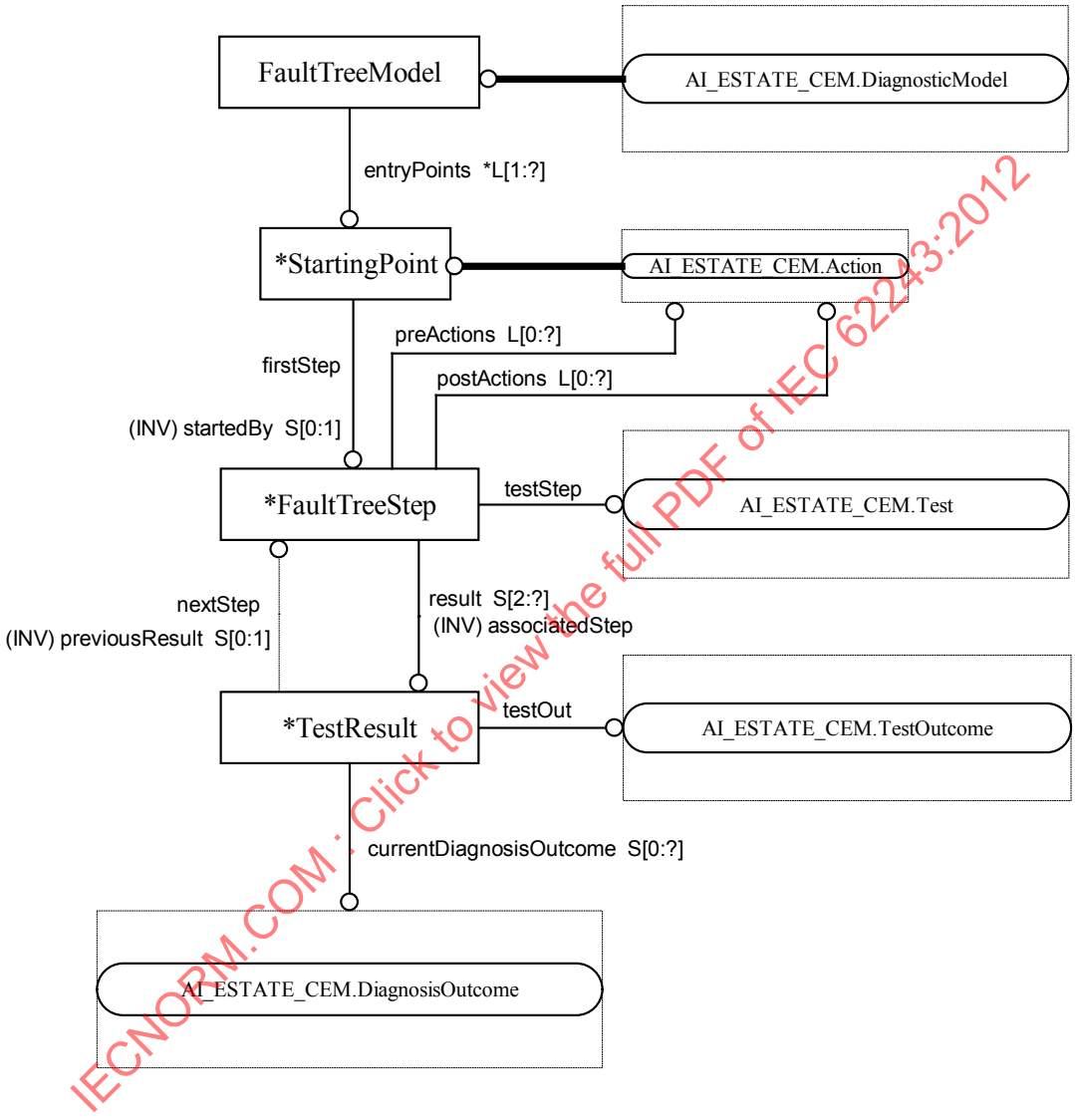


Figure 12—AI_ESTATE_FTM EXPRESS-G diagram 1 of 1

*)

(*

6.6 AI_ESTATE_DCM

The AI-ESTATE Dynamic Context Model information model captures a record of a reasoning session. The DCM enables the development of information interfaces to the historical results of AI-ESTATE reasoning

sessions, including the state of the reasoner and each step in the reasoning process. The DCM data and knowledge are developed during a diagnostic session, unlike those of the CEM, FTM, D-Matrix Inference Model, Diagnostic Logic Model, and Bayesian Network Model (BNM) (which consist of static diagnostic data and knowledge).

A diagnostic session is initiated by identifying the model to be used for determining the existence of a fault in the unit undergoing test and for isolating to a sufficient level to effect a maintenance action that will restore the system to a known functioning condition. The session proceeds in a series of steps. At each step, one or more tests are performed. The DCM is used to record the state existing prior to performing any test at each step as well as the results after performing the test. At each step, the following are recorded:

— The status of all actions, diagnoses, and resources at the start of the step.

— The actions that are performed.

— The outcomes of those actions.

— Optionally, the current fault hypothesis

The DCM schema references types from the CEM schema to enable DCM instances to record what transpired during a diagnostic session. The CEM entities in a DCM instance are copies of entities from the diagnostic model used during the session. For example, certain DCM entities point to the CEM entities to indicate "that Action was performed," and "that Outcome was observed," and "that DiagnosisOutcome was inferred."

A DCM instance contains an abridged copy of the diagnostic model instance that was used during the session, not a full copy. Since the DCM schema only references the CEM, a DCM instance can not contain instances of types declared in the BNM, DIM DLM or FTM even if one of those models types were used. In addition, a DCM instance can omit CEM entities that played no role in the session, or recast a subtype as a supertype as a way of pruning unnecessary diagnostic model information, and even omit or null unimportant attribute values from the static model. There is no need for a DCM instance to retain details of a static model that are not required for recording the session.

EXPRESS specification:

```
*)
SCHEMA AI_ESTATE_DCM;

   REFERENCE FROM AI_ESTATE_CEM;

(*
```

6.6.1 HypothesisDirected

Type HypothesisDirected defines a type by which a step can determine whether the search is focused on a provided user hypothesis or on a search process applied by the reasoner.

EXPRESS specification:

```
*)
   TYPE HypothesisDirected = BOOLEAN;
   END_TYPE;
(*
```

6.6.2 Identifier

Type Identifier defines a string type for specifying identification information.

EXPRESS specification:

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

6.6.3 ReliabilityDirected

Type ReliabilityDirected defines a type by which a step can determine whether or not the optimization process depends on failure distributions.

EXPRESS specification:

```
*)
   TYPE ReliabilityDirected = BOOLEAN;
   END_TYPE;
(*
```

6.6.4 TimeStamp

Type TimeStamp defines a string for representing date and time information in accordance with World Wide Web Consortium (W3C) XML Schema Part 2 [B4] Par. 3.2.7 dateTime definition.

EXPRESS specification:

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

6.6.5 URI

Type URI defines a string for specifying a uniform resource identifier in accordance with IETF RFC 2396-1998.

EXPRESS specification:

```
*)
```

```
   TYPE URI = STRING;
   END_TYPE;
(*
```

6.6.6 StatusCode

Enumerated type StatusCode specifies legal status codes to be returned by a service specified by this standard.

EXPRESS specification:

```
*)
   TYPE StatusCode = ENUMERATION OF
      (OPERATION_COMPLETED_SUCCESSFULLY,
      NONEXISTENT_DATA_ELEMENT_REQUESTED,
      MISSING_OR_INVALID_ARGUMENT,
      OPERATION_OUT_OF_SEQUENCE,
      INVALID_MODEL_SCHEMA,
      SERVICE_NOT_AVAILABLE,
      UNKNOWN_EXCEPTION_RAISED);
   END_TYPE;
(*
```

6.6.7 ActiveAction

Entity ActiveAction corresponds to an action that has been taken in the session. Active actions have actual costs that are tied back to the associated tests and resources.

EXPRESS specification:

```
*)
   ENTITY ActiveAction;
      actionType     : Action;
      costIncurred   : SET OF ActualCost;
   INVERSE
      stepPerformed  : Step FOR actionsPerformed;
   END_ENTITY;
(*
```

Attribute definitions:

actionType : Attribute actionType identifies the type of action performed as specified within the Common Element Model.

costIncurred : Attribute costIncurred provides the set of actual costs associated with taking this action in the current step of the session trace.

stepPerformed : Inverse attribute stepPerformed identifies the specific step within the session where the action was performed.

6.6.8 ActualCost

Entity ActualCost provides the cost information collected when an associated action is performed.

EXPRESS specification:

```
*)
   ENTITY ActualCost;
     actualValue    : CostValue;
     costType       : Cost;
   INVERSE
     actionCost     : ActiveAction FOR costIncurred;
   WHERE
     valueIsValid   : (NOT(EXISTS(costType.lowerBound)) OR
                       (costType.lowerBound <= actualValue)) AND
                      (NOT(EXISTS(costType.upperBound)) OR
                       (costType.upperBound >= actualValue));
     validCostType  : EXISTS(actionCost.actionType.hasCost) AND
                      (costType IN actionCost.actionType.hasCost);
   END_ENTITY;
(*
```

Attribute definitions:

actualValue : Attribute actualValue records the incurred cost value.

costType : Attribute costType identifies the type of cost incurred as specified within the Common Element Model.

actionCost : Inverse attribute actionCost identifies the specific action to which this cost belongs.

Formal propositions:

valueIsValid — Proposition valueIsValid ensures that actualValue lies between legal bounds, given the bounds have been defined.

validCostType — Proposition validCostType ensures that the type of cost associated with the ActualCost corresponds to one of the expected cost types from the diagnostic model.

6.6.9 ActualOutcome

Entity ActualOutcome provides information on specific outcomes in the diagnostic process.

EXPRESS specification:

```
*)
   ENTITY ActualOutcome;
     actualConfidence  : OPTIONAL ConfidenceValue;
     outcomeType       : Outcome;
   INVERSE
     stepInferred      : SET [0:?] OF Step FOR outcomesInferred;
     stepRecorded      : SET [0:1] OF Step FOR outcomesObserved;
     stepHypothesized  : SET [0:1] OF Step FOR userHypothesis;
   WHERE
     boundConfidence   : NOT(EXISTS(SELF.actualConfidence)) OR
```

```
                          (SELF.actualConfidence <=
                          SELF.outcomeType.maxConfidence);
    associatedStep    :   (SIZEOF(stepInferred)>=1) XOR
                          (SIZEOF(stepRecorded)=1) XOR
                          (SIZEOF(stepHypothesized)=1);
  END_ENTITY;
 (*
```

Attribute definitions:

actualConfidence	:	Attribute actualConfidence provides the specific confidence value associated with the actual outcome.
outcomeType	:	Attribute outcomeType identifies the corresponding Outcome defined in the Common Element Model.
stepInferred	:	Inverse attribute stepInferred identifies the step in the diagnostic session where the given inferred outcomes were recorded.
stepRecorded	:	Inverse attribute stepRecorded identifies the step in the diagnostic session where the current ActualOutcome was recorded.

Formal propositions:

boundConfidence	Proposition boundConfidence specifies that the actual confidence must be no more than the maximum confidence recorded in the Common Element Model for this outcome.
associatedStep	Proposition associatedStep specifies that the ActualOutcome must play a role as either an inferred outcome, an observed outcome, or a user hypothesis, but not more than one of these roles. An ActualOutcome that is an inference can be used in that role by multiple Steps to represent that the inference is unchanging from Step to Step in Session.trace. An ActualOutcome that is an observation or user hypothesis can only have that role in the one Step where the observation or hypothesis was asserted.

6.6.10 ActualSystemItem

Entity ActualSystemItem identifies the specific SystemUnderTest that is the focus of diagnosis in the current diagnostic session. It also provides a reference to any diagnostic history information on the unit or system under test in terms of previous test/diagnosis/repair sessions.

EXPRESS specification:

```
*)
  ENTITY ActualSystemItem;
    diagnosedItemID : Identifier;
    history         : SET OF URI;
    itemUnderTest   : SystemItem;
  INVERSE
    diagnosedIn     : Session FOR diagnosedItem;
  UNIQUE
    oneId     :     diagnosedItemID;
```

```
   END_ENTITY;
(*
```

Attribute definitions:

diagnosedItemID : Attribute diagnosedItemID provides a specific unique identifier for the system or unit being diagnosed.

history : Attribute history identifies a set of diagnostic sessions capturing historical diagnostic information.

itemUnderTest : Attribute itemUnderTest identifies the type of RepairItem that is the subject of diagnosis

diagnosedIn : Attribute diagnosedIn identifies the Session where the system was diagnosed.

6.6.11 ActualUsage

Entity ActualUsage provides information on time of use by a SystemItem. The purpose of the entity is to capture life cycle information for the purposes of determining where the unit lies within its reliability distribution. This information is associated with a specific SystemItem under the assumption that the information "flows down" to child SystemItems.

EXPRESS specification:

```
*)
   ENTITY ActualUsage;
     topLevel       : SystemItem;
     units          : TimeUnit;
     timeIndex      : TimeValue;
   INVERSE
     stepRecorded   : Step FOR lifeCycleStatus;
   END_ENTITY;
(*
```

Attribute definitions:

topLevel : Attribute topLevel identifies the top-most SystemItem to which the usage information applies.

units : Attribute units specifies the time units applied to the operational time of the system item.

timeIndex : Attribute timeIndex provides a value indicating the usage or operational time of the system item relative to the time baseline.

stepRecorded : Inverse attribute stepRecorded identifies the step in the diagnostic session where the usage data is recorded.

6.6.12 Discrepancy

Entity Discrepancy captures information on source events that initiated the current diagnostic session.

EXPRESS specification:

```
*)
   ENTITY Discrepancy;
     name                 :  NameType;
     description          :  DescriptionType;
   INVERSE
     sessionInitiated     :  SET [0:1] OF Session FOR cause;
   END_ENTITY;
(*
```

Attribute definitions:

name : Attribute name provides a unique name for identifying the trigger.

description : Attribute description provides a means for associating descriptive text to characterize the trigger.

sessionInitiated : Inverse attribute sessionInitiated identifies the diagnostic session that was initiated by the given discrepancy.

6.6.13 ServiceState

Entity ServiceState provides information on the status of a completed AI-ESTATE service request.

EXPRESS specification:

```
*)
   ENTITY ServiceState;
     associatedService    : DescriptionType;
     description          : DescriptionType;
     status               : StatusCode;
   INVERSE
     stepRecorded         : SET [0:1] OF Step FOR serviceLog;
   END_ENTITY;
(*
```

Attribute definitions:

associatedService : Attribute associatedService identifies the AI-ESTATE service called.

description : Attribute description provides a textual description for the AI-ESTATE service state.

status : Attribute status identifies the status code returned by an AI-ESTATE service.

stepRecorded : Inverse attribute stepRecorded identifies the step in the diagnostic session where the AI-ESTATE service state was recorded.

6.6.14 Session

Entity Session is the collector for the trace of steps performed in the current diagnostic session.

EXPRESS specification:

```
*)
  ENTITY Session;
    name                      : NameType;
    cause                     : OPTIONAL SET [1:?] OF Discrepancy;
    modelForDiagnosis         : DiagnosticModel;
    timeInitiated             : TimeStamp;
    trace                     : LIST OF Step;
    diagnosedItem             : ActualSystemItem;
  UNIQUE
    oneName   :               name;
  WHERE
    noTestsLast               : (SIZEOF(SELF.trace) = 0) OR
                                (SIZEOF(trace[SIZEOF(trace)].outcomesObser
                                ved) = 0);
    firstStepHasResources     : NOT(SIZEOF(SELF.trace) > 0) OR
                                EXISTS(SELF.trace[1].availableResources);
    commonSystem              : SELF.modelForDiagnosis.systemUnderTest =
                                SELF.diagnosedItem.itemUnderTest;
  END_ENTITY;
(*
```

Attribute definitions:

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

cause : Attribute cause identifies one or more discrepancies that initiated the diagnostic session.

modelForDiagnosis : Attribute modelForDiagnosis identifies the diagnostic model that was used in the current diagnostic session.

timeInitiated : Attribute timeInitiated identifies the specific time at which the diagnostic session was started.

trace : Attribute trace provides an ordered list of steps that the diagnostic reasoner performs during a session. It is expected that actions such as backing up would be recorded as a new step in the trace rather than having a step deleted from the trace.

diagnosedItem : Inverse attribute diagnosedItem identifies the specific unit or system being diagnosed in the session.

Formal propositions:

noTestsLast — Proposition noTestsLast ensures that the last step in the traces has no associated outcomes or tests that have just been observed. To have observed this information indicates the state must be updated, and a new step must be created.

firstStepHasResources — Proposition firstStepHasResources requires that the first step in the session trace shall instantiate the availableResources attribute.

commonSystem — Proposition commonSystem ensures that the itemUnderTest identified as the ActualSystemItem corresponds to the systemUnderTest of the DiagnosticModel being used for diagnosis.

6.6.15 Step

Entity Step is the collector for the information recorded at each step in the current diagnostic session.

EXPRESS specification:

```
*)
  ENTITY Step;
    name                          : OPTIONAL NameType;
    actionsPerformed              : LIST OF ActiveAction;
    optimizedByCost               : SET OF CostCategory;
    optimizedByDistribution       : ReliabilityDirected;
    optimizedByUser               : HypothesisDirected;
    outcomesInferred              : SET OF ActualOutcome;
    outcomesObserved              : SET OF ActualOutcome;
    serviceLog                    : OPTIONAL LIST OF ServiceState;
    stepContext                   : OPTIONAL ContextState;
    timeOccurred                  : OPTIONAL TimeStamp;
    userHypothesis                : OPTIONAL SET [1:?] OF ActualOutcome;
    lifeCycleStatus               : LIST OF ActualUsage;
    reverted                      : OPTIONAL Step;
    availableResources            : OPTIONAL SET [1:?] OF Resource;
    INVERSE
    owningSession                 : Session FOR trace;
  UNIQUE
    oneName   :                   name;
  WHERE
    hypothesisWithUserDirected    : (NOT(SELF.optimizedByUser) OR
                                     EXISTS(SELF.userHypothesis));
  END_ENTITY;
(*
```

Attribute definitions:

name : Attribute name provides an optional, unique name to be used when setting or restoring waypoints in a diagnostic session.

actionsPerformed : Attribute actionsPerformed identifies the ordered list of actions (e.g., tests, repairs, or other general actions) that have been performed at this step in the process.

optimizedByCost : Attribute optimizedByCost identifies the set of cost criteria (if any) used to optimize test selection at a given step in the diagnostic process.

optimizedByDistribution : Attribute optimizedByDistribution specifies that test selection is dependent on failure distribution when this Boolean attribute is TRUE.

optimizedByUser : Attribute optimizedByUser is set to TRUE when reasoning at a particular step was based on a user hypothesis and FALSE when based on the reasoner's hypothesis.

outcomesInferred : Attribute outcomesInferred identifies the inferred outcome values and confidences for all model outcomes, whether new or not, at this step.

outcomesObserved : Attribute outcomesObserved provides the set of test, diagnosis, and action outcomes observed at this step in the process, along with their associated confidence values.

serviceLog : Attribute serviceLog provides the list of status values returned as a result of executing an AI-ESTATE service.

stepContext : Attribute stepContext provides context information from the Common Element Model such as the system and operational state, as well as the reason for diagnosis at this step. The context may change as diagnosis proceeds, and once a context is set for a given step, that context will persist through subsequent steps in a session until changed.

timeOccurred : Attribute timeOccurred records a time stamp at which the step began.

userHypothesis : Attribute userHypothesis identifies a set of user-provided outcomes taken from the diagnostic model to be used as a working hypothesis by the reasoner. Note this is an optional attribute because a user may not have a hypothesis at all steps in the process (if ever).

lifeCycleStatus : Attribute lifeCycleStatus provides usage information for the system items within the system being diagnosed. The attribute is defined as a list to impose an order of precedence in the event a specific system item has multiple parents. Should this occur, later parents in the list will take precedence (i.e., will override) earlier parents.

reverted : Attribute reverted identifies the step returned to either through backtrack or restoreWaypoint. This attribute is populated when restoring the current Step. Otherwise, the attribute is not populated.

availableResources : Attribute availableResources identifies the resources that are available at a particular Step in the diagnostic session. If this attribute does not exist on a particular Step, then the resources from the last Step that instantiates the attribute are still available; however, if the attribute exists, then it shall enumerate all available resources at that Step. The first Step in the session shall instantiate this attribute.

owningSession : Inverse attribute owningSession identifies the specific session trace to which this step belongs.

Formal propositions:

hypothesisWithUserDirected Proposition hypothesisWithUserDirected specifies that if hypothesis-directed search is used at a step in the diagnosis (optimizedByUser = TRUE), then userHypothesis must exist.

6.6.16 ModelRule

Rule ModelRule shall apply to the population of ActualSystemItem entities in a DCM exchange file.

EXPRESS specification:

```
*)
   RULE ModelRule FOR
      (ActualSystemItem);
   WHERE
      oneModel  : SIZEOF(ActualSystemItem) = 1;
   END_RULE;
END_SCHEMA;
(*
```

6.6.17 Dynamic Context Model EXPRESS-G diagrams

The EXPRESS-G definition of the DCM is represented by Figure 13 through Figure 15.

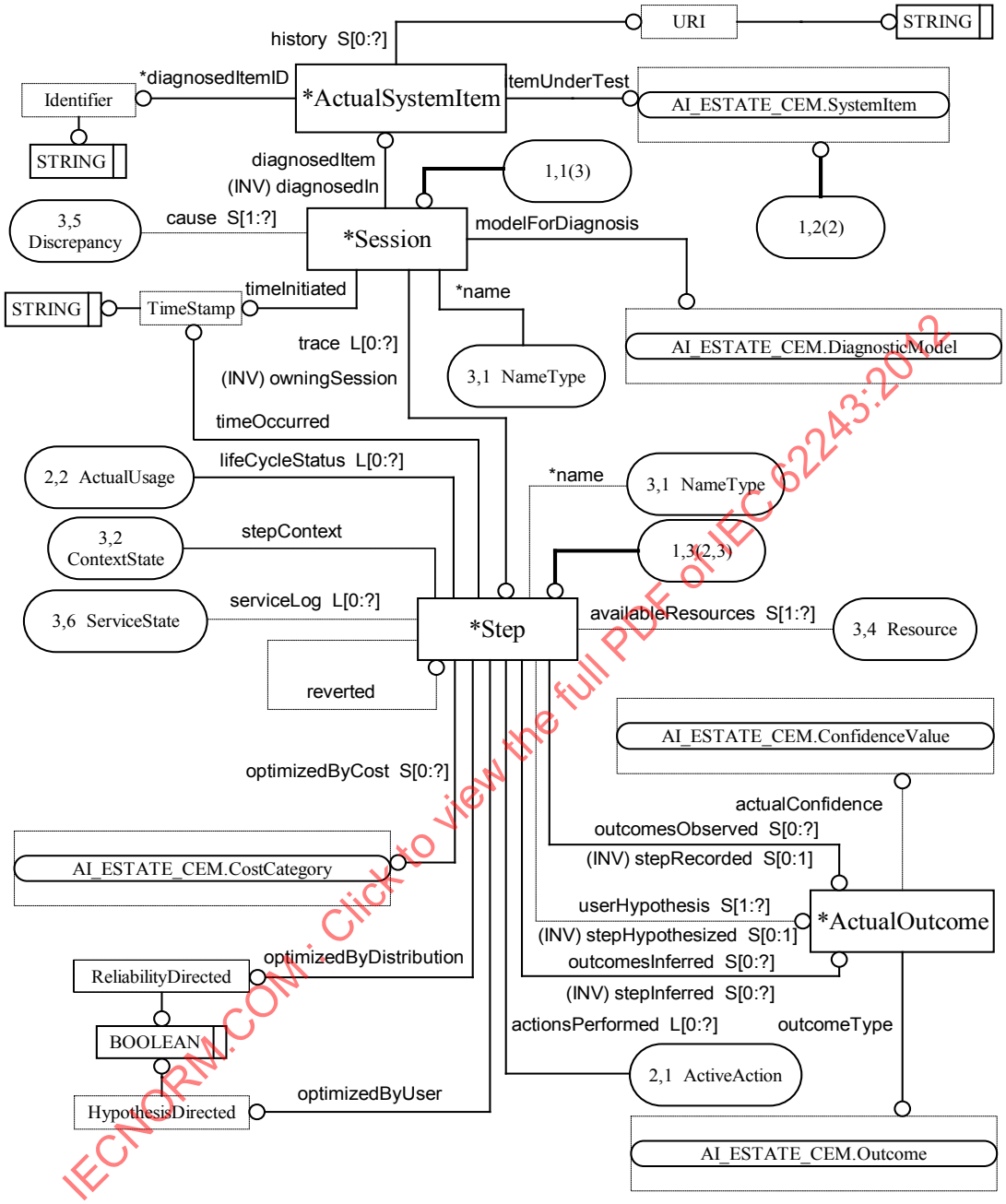


Figure 13—AI_ESTATE_DCM EXPRESS-G diagram 1 of 3

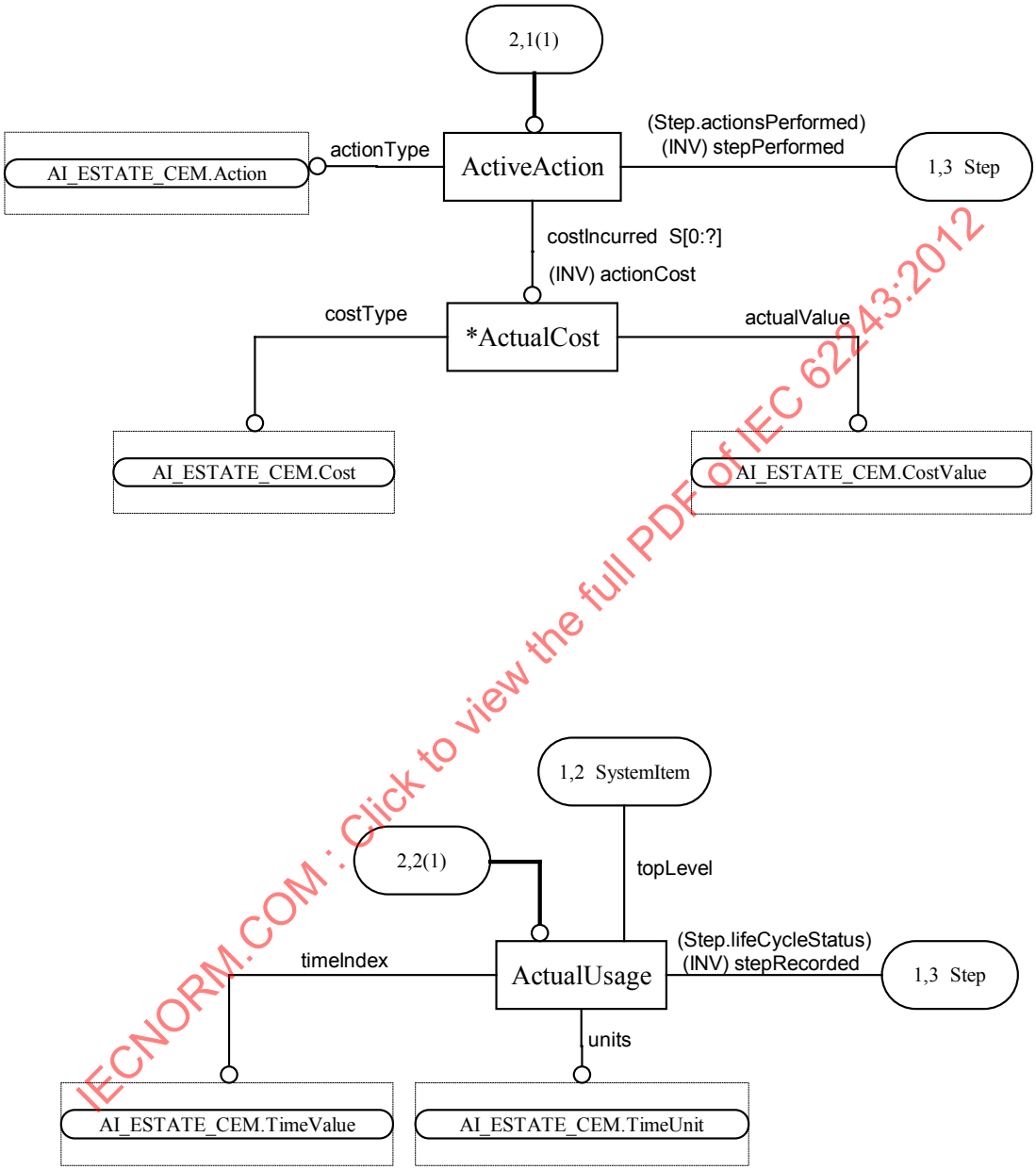


Figure 14—AI_ESTATE_DCM EXPRESS-G diagram 2 of 3

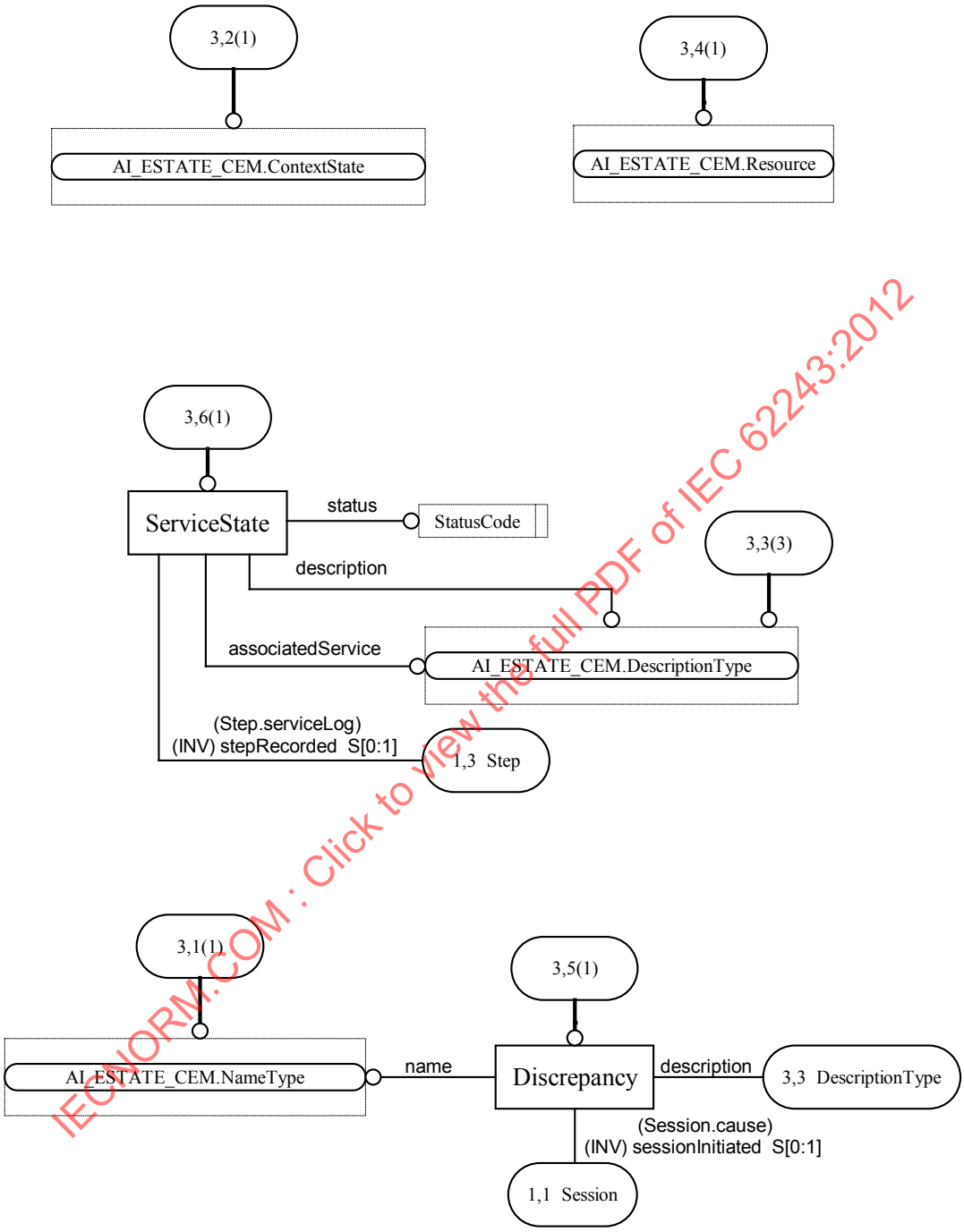


Figure 15—AI_ESTATE_DCM EXPRESS-G diagram 3 of 3

*)

(*

7. Reasoner manipulation services

This clause specifies a set of reasoner manipulation services for interacting with a diagnostic reasoner application during a diagnostic session. A conformant diagnostic reasoner shall expose these services for a client application to call. The services provide an interface that allows the reasoner to receive test information from the client, provide diagnostic conclusions and other analysis to the client, recommend actions to the client, and create a record of the session. The role of the client application is to control the diagnostic process, provide test information to the reasoner, request diagnostic conclusions, and perform test and repair actions. Within a diagnostic session, the client application will typically call the services in this clause to do the following:

- Tell the reasoner to start a new diagnostic session
- Tell the reasoner which static model to use
- Get diagnostic conclusions and recommended actions from the reasoner
- Tell the reasoner the outcomes of actions the client performed
- Tell the reasoner to save a record of the session
- End the session

The reasoner manipulation services in this clause are designed to be sufficient for a typical diagnostic session. Any extensions to the reasoner manipulation services shall adhere to 5.2.

This clause specifies services using EXPRESS notation; this in no way restricts the software language one uses for implementing the services. In addition, several services input or output service-specific EXPRESS entities that are defined within the service specification. Where an EXPRESS entity is defined as part of a service specification, the entity definition specifies the *information content* of the input or output; not the implementation *format*. The implementation format is user defined and should adhere to the description in 4.2. For simplicity, the entities are defined such that there are no explicit entity-to-entity relationships within a service call or between services calls. That is, no entity defined in this clause has another entity as an attribute.

The reasoner manipulation services in this clause make frequent use of “NameType” attributes on entities in the static diagnostic model. Names are permanent identifiers for items understood on both sides of the reasoner-client interface. For example, when the reasoner recommends that a test be performed, it does so by passing the test name to the client. The client application should recognize that test name and be able to execute the corresponding test procedure. Similarly, the client informs the reasoner that it has performed a test by passing the test name, and the reasoner recognizes the test name.

7.1 Service order dependence

This subclause specifies the order dependencies for Reasoner Manipulation Services by defining a set of states for the reasoner application, the services available during each state, and the state transitions that result from certain services. The basic execution model for an AI-ESTATE conformant diagnostic reasoner shall manipulate an instance of the DCM to maintain reasoner state and follows the high-level state diagram shown in Figure 16. The five states defined in the state diagram are described in Table 1.

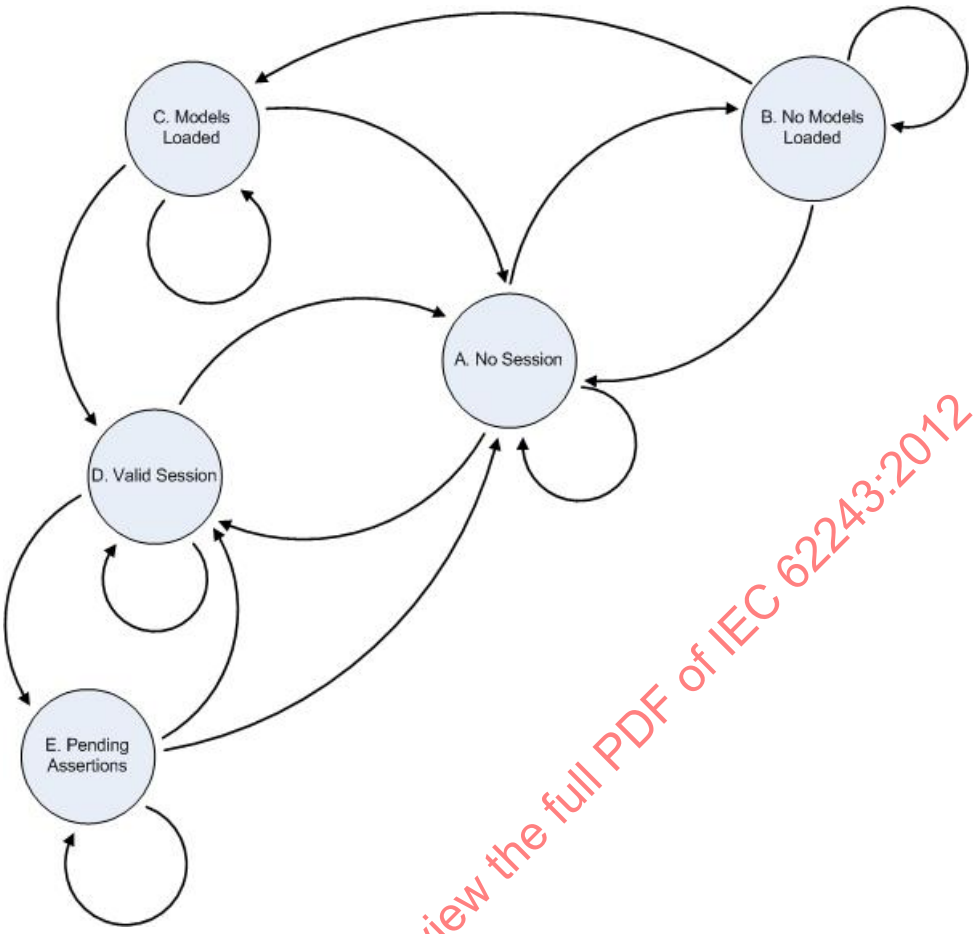


Figure 16—Execution model for an AI-ESTATE conformant diagnostic reasoner

Table 1—State descriptions

State	State descriptions
A. No Session	"No Session" is where the reasoner application can start a new session for a particular unit under test (UUT). A reasoner's state shall be state A at startup. The reasoner shall successfully transition to state A from any state when the service closeDiagnosticProcess is called.
B. No Models Loaded	"No Models Loaded" is the state where no DiagnosticModels are yet loaded and where they are loaded. This state also permits informing the reasoner of any historical data for the UUT and the discrepancy that initiated the current session.
C. Models Loaded	"Models Loaded" exists to ensure that at least one DiagnosticModel has been loaded before transitioning to state D and supports services that require models be present.
D. Valid Session	"Valid Session" supports client requests for information from the reasoner. The reasoner is in state D after it instantiates a new step in the session trace. The reasoner stays in state D so long as the client only asks for information. The reasoner transitions out of state D if the client asserts new information to the reasoner.
E. Pending Assertions	"Pending Assertions" supports asserting information to the reasoner. The reasoner transitions to state E after the client asserts information. The reasoner stays in state E so long as the client only asserts information. The reasoner transitions out of state E if the client asks for information from the reasoner.

States D and E are essentially indistinguishable from the client's perspective, as the same services are available in both states. The distinction between states D and E relates to the DCM, particularly whether the service instantiates a new step in the session trace.

Each state is represented by a corresponding "state column" in the following table. Transitions between states (e.g., A to B, C to D) are accomplished by executing the services associated with the relevant columns (labeled **AB**, **CD**, etc.) The rules for remaining in the current step, instantiating a new step, or reverting to an earlier step are specified by the numbered cell entries in the table and the numbered list of rules below the table. The rules also specify services' effects on the current step as recorded in the DCM.

Each service has a row in the table. Grayed out cells indicate that the service is unavailable in that state, such that the reasoner shall raise an "OPERATION_OUT_OF_SEQUENCE" exception if the service is called from that state. Each state column has several minor columns denoting state transitions. The heading at the top of the minor column defines an initial and final state. For example, DE indicates a transition from state D to E, whereas DD indicates the reasoner stays in state D. An entry in a cell in the minor column indicates that a call to the service will result in the corresponding state transition if the call completes successfully. The reasoner shall not make the state transition if the service completed unsuccessfully.

Table 2 defines the state transitions for the Reasoner Manipulation Services. A conformant diagnostic reasoner shall support sequences of calls to Reasoner Manipulation Services in any order allowed by the table. The reasoner shall raise a "OPERATION_OUT_OF_SEQUENCE" exception when the client calls a service that is out of sequence according to the table.

Table 2—Reasoner manipulation services state transitions

State A No session			State B No model loaded			State C Model loaded			State D Valid session			State E Pending assertions				
AA	AB	AD	BA	BB	BC	CA	CC	CD	DA	DD	DE	EA	ED	EE	Clause	Service
	X														7.4.7	initializeDiagnosticProcess
		X													7.4.14	restoreCheckpoint
		X													7.4.16	resumeDiagnosticProcess
X			X			X			X			X			7.4.4	closeDiagnosticProcess
X				X			X			2				2	7.4.5	describeReasoner
				X			X								7.4.10	loadHistoryFromLocation
				X			X								7.4.23	setDiscrepancies
					X		X								7.4.8	loadDiagnosticModel
					X		X								7.4.9	loadDiagnosticModelFromLocation
								X							7.4.19	setActiveModel
										1			1		7.4.27	updateState
										1			1		7.4.22	setContext
										2			4		7.5.1	estimatedCostsToStage
										2			4		7.5.2	estimatedResourcesToStage
										2			4		7.4.6	getDiagnosticResults
										2			4		7.5.3	getTestOutcomesFromDiagnosisOutcome
										2			4		7.4.12	recommendActions
										2			4		7.4.13	requestResourcesNeeded
										2				2	7.4.26	showSessionTrace
										2			4		7.4.21	setCheckpoint
										2			4		7.4.24	setWaypoint
										3			3		7.4.15	restoreWaypoint
										X			X		7.4.3	backtrack
											2			2	7.4.2	applyDiagnosticOutcomes
											2			2	7.4.1	applyActions
											2			2	7.4.25	setUsage
											2			2	7.4.20	setAvailableResources

NOTE—Rules for states D and E:

a) Record service call. Close current step. Append new step and populate attributes. Any service input parameters apply to the attributes of the new step per the service specification.

b) Stay in current step. Record service call. Any service input parameters apply to the attributes of the current step per the service specification.

c) Record service call in the current step. Append a new step that duplicates the specified the Step (possibly the current step) as it was in its intial state before the first service call occurred in that Step.

d) Equivalent to calling updateState() followed by this service.

7.2 Status codes

An AI-ESTATE diagnostic reasoner shall provide a means for its clients to determine the success or failure of service requests. This shall be recorded by means of the ServiceState entity within the Dynamic Context Model. This entity will record one of the following status codes:

- OPERATION_COMPLETED_SUCCESSFULLY
- NONEXISTENT_DATA_ELEMENT_REQUESTED
- MISSING_OR_INVALID_ARGUMENT
- OPERATION_OUT_OF_SEQUENCE
- INVALID_MODEL_SCHEMA
- SERVICE_NOT_AVAILABLE
- UNKNOWN_EXCEPTION_RAISED

The status code usage is described in the service definitions.

7.3 Data types for the reasoner manipulation services

The AI-ESTATE Reasoner Service Model information model specifies information elements and their types for use by the services defined within the AI-ESTATE standard. The entities and types specified in this model are intended to be used by the services and not as a means of data exchange through either the ISO 10303-21:1994 or the ISO 10303-28:2007 interfaces.

EXPRESS specification:

```
*)
SCHEMA AI_ESTATE_RSM;

   REFERENCE FROM AI_ESTATE_CEM;

   REFERENCE FROM AI_ESTATE_DCM;

(*
```

7.3.1 ActionOutcomeValue

Type ActionOutcomeValue defines the subset of OutcomeValues that are valid for an ActionOutcome.

EXPRESS specification:

```
*)
   TYPE ActionOutcomeValue = OutcomeValues;
   WHERE
      valid   : SELF IN [NOTSTARTED, COMPLETED, ABORTED, NOTKNOWN,
                NOTAVAILABLE, USERDEFINED];
   END_TYPE;
(*
```

Formal propositions:

valid Proposition valid ensures the value corresponds to a legal ActionOutcome value.

7.3.2 DiagnosisOutcomeValue

Type DiagnosisOutcomeValue defines the subset of OutcomeValues that are valid for a DiagnosticOutcome.

EXPRESS specification:

```
*)
  TYPE DiagnosisOutcomeValue = OutcomeValues;
  WHERE
    valid    : SELF IN [GOOD, BAD, CANDIDATE, NOTKNOWN, USERDEFINED];
  END_TYPE;
(*
```

Formal propositions:

valid Proposition valid ensures the outcome value is one of the legal values given by DiagnosisOutcome.

7.3.3 TestOutcomeValue

Type TestOutcomeValue defines the subset of OutcomeValues that are valid for a TestOutcome.

EXPRESS specification:

```
*)
  TYPE TestOutcomeValue = OutcomeValues;
  WHERE
    valid   : SELF IN [PASS, FAIL, NOTKNOWN, USERDEFINED];
  END_TYPE;
(*
```

Formal propositions:

valid Proposition valid ensures the value is a legal TestOutcome value.

7.3.4 ExchangeFormat

Enumerated type ExchangeFormat enumerates the exchange file formats for diagnostic models and dynamic context model instances.

P21 : The ISO 10303-21 exchange format specified in 5.1.1.

P28 : The ISO 10303-28 exchange format specified in 5.1.2.

NATIVE : An implementation-specific format.

EXPRESS specification:

```
*)
   TYPE ExchangeFormat = ENUMERATION OF
      (P21,
      P28,
      NATIVE);
   END_TYPE;
(*
```

7.3.5 OutputType

Enumerated type OutputType enumerates the types of diagnostic conclusions that the calling application can request from the reasoner. The reasoner shall be capable of returning each of these types, provided the requisite information is present in the active diagnostic model.

NO_FAULT_RESULT : The inferred state of the NoFault Diagnosis. The reasoner shall be capable of returning this result regardless of whether a Diagnosis with the name NoFault is literally instantiated in the active diagnostic model.

DIAGNOSIS_RESULT : The inferred state of Diagnosis instances in the active diagnostic model, excluding subtypes of Diagnosis, and excluding the special Diagnosis with the name "No Fault".

FAULT_RESULT : The inferred state of Fault instances in the active diagnostic model.

FAILURE_RESULT : The inferred state of Failure instances in the active diagnostic model.

REPAIR_ITEM_RESULT : The inferred state of RepairItem instances in the active diagnostic model.

FUNCTION_ITEM_RESULT : The inferred state of FunctionItem instances in the active diagnostic model.

EXPRESS specification:

```
*)
   TYPE OutputType = ENUMERATION OF
      (NO_FAULT_RESULT,
      DIAGNOSIS_RESULT,
```

```
      FAULT_RESULT,
      FAILURE_RESULT,
      REPAIR_ITEM_RESULT,
      FUNCTION_ITEM_RESULT,
      ALL);
   END_TYPE;
(*
```

7.3.6 StageType

Enumerated type StageType enumerates the stages of a diagnostic session in the order that they occur. The thresholds for meeting these stages are implementation specific. Some implementations may not distinguish between some of these stages, but they shall preserve the order.

DETECTED : The initial determination that there must be a fault/failure (e.g., the first test to fail).

ISOLATED : The point in the process where repair becomes a reasonable action.

REPAIRED : The point in the process where the proper repair action(s) have been taken.

VERIFIED : The point in the process where any problems are confirmed to be corrected. For the no-fault-found case, the point in the process where no-fault-found is confirmed.

FINISHED : The point in the process where the last reasoner-directed action is complete.

EXPRESS specification:

```
*)
   TYPE StageType = ENUMERATION OF
      (DETECTED,
      ISOLATED,
      REPAIRED,
      VERIFIED,
      FINISHED);
   END_TYPE;
(*
```

7.3.7 CostUnit

Select type CostUnit combines the TimeUnit and NonTimeUnit types to form a generalized unit of cost.

EXPRESS specification:

```
*)
   TYPE CostUnit = SELECT
      (TimeUnit,
      NonTimeUnit);
   END_TYPE;
(*
```

7.3.8 ActionRecommendation

Entity ActionRecommendation represents a single action or a sequence of actions that the reasoner recommends taking.

EXPRESS specification:

```
*)
  ENTITY ActionRecommendation;
    actionNames            :  LIST [1:?] OF NameType;
    actionDescriptions     :  LIST [1:?] OF DescriptionType;
    sequenceDescription    :  DescriptionType;
    costCategories         :  LIST OF NameType;
    categoryDescriptions   :  LIST OF DescriptionType;
    estimates              :  LIST OF CostValue;
    uppers                 :  LIST OF CostValue;
    lowers                 :  LIST OF CostValue;
    units                  :  LIST OF CostUnit;
    contextNames           :  LIST OF SET OF NameType;
    contextDescriptions    :  LIST OF SET OF DescriptionType;
  WHERE
    actionTable     : SIZEOF(actionNames)=SIZEOF( actionDescriptions);
    costTable :     (SIZEOF(costCategories) =
                      SIZEOF(categoryDescriptions)) AND
                      (SIZEOF(costCategories) = SIZEOF(estimates)) AND
                      (SIZEOF(costCategories) = SIZEOF(uppers)) AND
                      (SIZEOF(costCategories) = SIZEOF(lowers)) AND
                      (SIZEOF(costCategories) = SIZEOF(units));
    alignedContext  : SIZEOF(contextNames) IN [0, SIZEOF(actionNames)];
    validContext    : (SIZEOF(contextNames)=
                      SIZEOF(contextDescriptions));
  END_ENTITY;
(*
```

Attribute definitions:

actionNames : Attribute actionNames lists the Action or sequence of Actions by name.

actionDescriptions : Attribute actionDescriptions lists the description attribute of each Action in actionNames. The list actionDescriptions is aligned with actionNames.

sequenceDescription : Attribute sequenceDescription provides a description for a sequence of actions. Omit sequenceDescription if only one action is in the actionNames list.

costCategories : Attribute costCategories provides a list of names of the CEM CostCategory that are associated with the Actions specified by actionNames.

categoryDescriptions : Attribute categoryDescriptions lists the description attribute value of each CostCategory in costCategories. The list categoryDescriptions is aligned with costCategories.

estimates : Attribute estimates lists the reasoner estimates of each cost in costCategories. The list estimates is aligned with costCategories.

uppers : Attribute uppers lists the upper bounds of the reasoner estimates of each cost in costCategories. The list uppers is aligned with costCategories.

lowers : Attribute lowers lists the lower bounds of the reasoner estimates of each cost in costCategories. The list lowers is aligned with costCategories.

units : Attribute units lists the unit designations used to characterize each cost value in costCategories. The list units is aligned with costCategories.

contextNames : Attribute contextNames provides the names of the ContextStates under which the client needs to perform each Action in actionNames. The outer list of contextNames is aligned with actionNames. The inner list forms a disjoint set, any single ContextState is acceptable for performing the Action. The inner list shall be a subset of Action.mustOccurIn. If the inner set is empty, the Action can be performed under any ContextState.

contextDescriptions : Attribute contextDescriptions provides the description attribute of each ContextState in contextNames. The outer list and inner set of contextDescriptions are aligned with contextNames.

Formal propositions:

actionTable — Proposition actionTable ensures that the lists actionNames and actionDescriptions are the same length.

costTable — Proposition costTable ensures that the lists costCategories, categoryDescriptions, estimates, uppers, lowers, and units are all the same length.

alignedContext — Proposition alignedContext ensures that the outer list of contextNames is the same length as actionNames, or that contextNames is empty.

validContext — Attribute validContext ensures that the outer list of contextDescriptions is the same length as the outer list of contextNames. The lengths of the inner lists shall also be equal, but this is not checked by the schema.

7.3.9 ActualAction

Entity ActualAction provides information on an action that was performed or attempted, any outcome, and any costs incurred.

EXPRESS specification:

```
*)
  ENTITY ActualAction;
    actionName         : NameType;
    statusValue        : OPTIONAL ActionOutcomeValue;
    statusConfidence   : OPTIONAL ConfidenceValue;
    costLabels         : OPTIONAL LIST OF NameType;
    costValues         : OPTIONAL LIST OF CostValue;
  WHERE
    costAligned       : (NOT(EXISTS(costLabels)) AND
                         NOT(EXISTS(costValues))) OR (SIZEOF(costLabels)
                         = SIZEOF(costValues));
```

```
    validConfidence :NOT(EXISTS(statusConfidence)) OR
                        EXISTS(statusValue);
  END_ENTITY;
(*
```

Attribute definitions:

actionName	:	Attribute actionName indicates the Action by name.
statusValue	:	Attribute statusValue is the status of the action, as specified by type ActionOutcomeValue. actionName combined with statusValue point to a ActionOutcome in the diagnostic model.
statusConfidence	:	Attribute statusConfidence is the confidence in the status value.
costLabels	:	Attribute costLabels provides a list of the names of the Costs incurred performing the Action. The costLabels point to CostCategory entities by name.
costValues	:	Attribute costValues provides the corresponding numerical values of the costs in costLabels. The list costValues is aligned with costLabels.

Formal propositions:

costAligned	Proposition costAligned ensures that the sizes of costLabels and costValues matches so the values can be aligned.
validConfidence	Proposition validConfidence ensures that a statusConfidence is only present if statusValue is present. statusValue can be present with or without statusConfidence.

7.3.10 ActualDiagnosisOutcome

Entity ActualDiagnosisOutcome reports data for a diagnostic outcome that was determined through some means other than via the current reasoned being used in the current diagnostic session.

EXPRESS specification:

```
*)
  ENTITY ActualDiagnosisOutcome;
    diagnosisName       : NameType;
    outcomeValue        : DiagnosisOutcomeValue;
    outcomeConfidence   : OPTIONAL ConfidenceValue;
  END_ENTITY;
(*
```

Attribute definitions:

diagnosisName	:	Attribute diagnosisName provides the name of the diagnosis whose outcome is being applied.

outcomeValue : Attribute outcomeValue provides the outcome of the diagnosis (GOOD, BAD, etc.).

outcomeConfidence : Attribute testOutcomeConfidence is the confidence in the outcomeValue.

7.3.11 ActualTest

Entity ActualTest is a subtype of ActualAction. It provides information on a Test that was performed or attempted.

EXPRESS specification:

```
*)
   ENTITY ActualTest
      SUBTYPE OF(ActualAction);
        testOutcomeValue        : TestOutcomeValue;
        testOutcomeQualifier    : OPTIONAL QualifierType;
        testOutcomeConfidence   : OPTIONAL ConfidenceValue;
   END_ENTITY;
(*
```

Attribute definitions:

testOutcomeValue : Attribute testOutcomeValue provides the observed outcome of the Test (PASS, FAIL, etc.).

testOutcomeQualifier : Attribute testOutcomeQualifier provides the optional qualifier of the observed test outcome. The combination of testOutcomeValue and testOutcomeQualifier identifies the particular TestOutcome entity that was observed.

testOutcomeConfidence : Attribute testOutcomeConfidence is the confidence in the observation.

7.3.12 ActualUsageData

Entity ActualUsageData represents the actual usage of a SystemItem. ActualUsageData is similar to entity ActualOutcome in the DCM, except it is tailored for use in the services.

EXPRESS specification:

```
*)
   ENTITY ActualUsageData;
      topLevelSystemItem   : NameType;
      units                : TimeUnit;
      timeIndex            : TimeValue;
   END_ENTITY;
(*
```

Attribute definitions:

topLevelSystemItem : Attribute topLevelSystemItem provides the name of the top-most SystemItem to which the usage information applies.

units : Attribute units specifies units of usage.

timeIndex : Attribute timeIndex provides the value of usage or operational time of the RepairItem relative to the time baseline.

7.3.13 CostEstimate

Entity CostEstimate corresponds to a projected Cost.

EXPRESS specification:

```
*)
  ENTITY CostEstimate;
    name         :  NameType;
    description  :  DescriptionType;
    estimate     :  CostValue;
    upper        :  CostValue;
    lower        :  CostValue;
    unit         :  CostUnit;
  END_ENTITY;
(*
```

Attribute definitions:

name : Attribute name points to a CostCategory in the DiagnosticModel via its name.

description : Attribute description is a copy of that CostCategory description attribute.

estimate : Attribute estimate is the expectation value for the cost.

upper : Attribute upper is the nominal upper bound of the cost.

lower : Attribute lower is the nominal lower bound of the cost.

unit : Attribute unit is the unit of measure for the cost.

7.3.14 DiagnosticConclusion

Entity DiagnosticConclusion corresponds to an inferred diagnostic conclusion from the reasoner. It supports any of the types of diagnostic conclusion defined by the type OutputType. It is similar to ActualOutcome in the DCM.

EXPRESS specification:

```
*)
  ENTITY DiagnosticConclusion;
```

```
    resultType    : OutputType;
    name          : NameType;
    description   : DescriptionType;
    outcome       : DiagnosisOutcomeValue;
    confidence    : ConfidenceValue;
  WHERE
    noFaultCase   : NOT((resultType=NO_FAULT_RESULT) XOR
                    (name=NoFault));
  END_ENTITY;
(*
```

Attribute definitions:

resultType : Attribute resultType indicates the type of the conclusion, as enumerated by type OutputType.

name : Attribute name provides the name of the SystemItem or Diagnosis to which this conclusion applies.

description : Attribute description contains a copy of the description attribute of the SystemItem or Diagnosis.

outcome : Attribute outcome is the inferred outcome value (GOOD, BAD, etc.) for the SystemItem or Diagnosis.

confidence : Attribute confidence is as follows:

For SystemItems and Diagnoses excluding NoFault, the following is true:

If outcome=GOOD, confidence is the reasoner's degree of certainty that the SystemItem or Diagnosis is indeed GOOD. If outcome=BAD, confidence is the reasoner's degree of certainty that the SystemItem or Diagnosis is indeed BAD.

If outcome=CANDIDATE, confidence is the reasoner's degree of certainty that the SystemItem or Diagnosis might be BAD.

If outcome=NOTKNOWN, confidence is the reasoner's degree of certainty that the SystemItem or Diagnosis might be BAD.

For the NoFault Diagnosis, the following is true:

If outcome=GOOD, confidence is the reasoner's degree of certainty that the SystemItem or Diagnosis is indeed GOOD (i.e., the SUT is fault-free). If outcome=BAD, confidence is the reasoner's degree of certainty that the SystemItem or Diagnosis is indeed BAD. See 7.3.2 for the semantics of GOOD, BAD, CANDIDATE, and NOTKNOWN for the NoFault.

If outcome=CANDIDATE, confidence is the reasoner's degree of certainty that the SystemItem or Diagnosis might be GOOD.

If outcome=NOTKNOWN, confidence is the reasoner's degree of certainty that the SystemItem or Diagnosis might be GOOD.

Formal propositions:

noFaultCase Proposition noFaultCase ensures that resultType=NO_FAULT_RESULT and name = NoFault appear together.

7.3.15 DiscrepancyData

Entity DiscrepancyData reports information on source events that initiated the current diagnostic session. It contains the information needed to instantiate a Discrepancy in the DCM.

EXPRESS specification:

```
*)
   ENTITY DiscrepancyData;
      name          : NameType;
      description   : DescriptionType;
   END_ENTITY;
(*
```

Attribute definitions:

name : Attribute name provides the name for the Discrepancy.

description : Attribute description provides the description for the Discrepancy.

7.3.16 ExpectedTestOutcome

Entity ExpectedTestOutcome provides the expected results of a Test if it were to be performed.

EXPRESS specification:

```
*)
   ENTITY ExpectedTestOutcome;
      name                    : NameType;
      outcomeValue            : TestOutcomeValue;
      outcomeQualifier        : OPTIONAL QualifierType;
      outcomeConfidence       : OPTIONAL ConfidenceValue;
      actionStatusValue       : OPTIONAL ActionOutcomeValue;
      actionStatusConfidence  : OPTIONAL ConfidenceValue;
   END_ENTITY;
(*
```

Attribute definitions:

name : Attribute name indicates the test by name.

outcomeValue : Attribute outcomeValue provides the test outcome value of the test (PASS, FAIL, etc.).

outcomeQualifier : Attribute outcomeQualifier provides the optional qualifier of the test outcome. The combination of outcomeValue and outcomeQualifier

points to a particular TestOutcome in the active diagnostic model.

outcomeConfidence : Attribute outcomeConfidence is the confidence that the outcomeValue and outcomeQualifier would be observed.

actionStatusValue : Attribute actionStatusValue provides the action outcome value of the test action. Use "UNAVAILABLE" if the test would be unavailable. Use completed if the test would PASS or FAIL.

actionStatusConfidence : Attribute actionStatusConfidence is the confidence that actionStatusValue would occur.

7.3.17 ResourceNeeded

Entity ResourceNeeded identifies a resource that is required to support the diagnostic process.

EXPRESS specification:

```
*)
   ENTITY ResourceNeeded;
      resourceName         : NameType;
      description          : DescriptionType;
      confidence           : ConfidenceValue;
      neededQuantity       : NUMBER;
      availableQuantity    : NUMBER;
   END_ENTITY;
(*
```

Attribute definitions:

resourceName : Attribute resourceName points to a Resource in the DiagnosticModel via its name.

description : Attribute description is a copy of that Resource's description attribute.

confidence : Attribute confidence quantifies the probability/confidence that the resource will be needed at the given neededQuanity.

neededQuantity : Attribute neededQuantity quantifies the amount of the resource that will be needed. Implementations that do not track quantity should use the integer 1.

availableQuantity : Attribute availableQuantity quantifies the availability of the resource as the reasoner knows it. Values>0 indicate the available quantity of the resource. The value=0 indicates it is unavailable. The values<0 indicate unknown availability. Implementations that do not track quantity should use the integer 1 to indicate the resource is available.

7.3.18 Reasoner Service Model EXPRESS-G diagrams

The EXPRESS-G definition of the RSM is represented by Figure 17 through Figure 19.

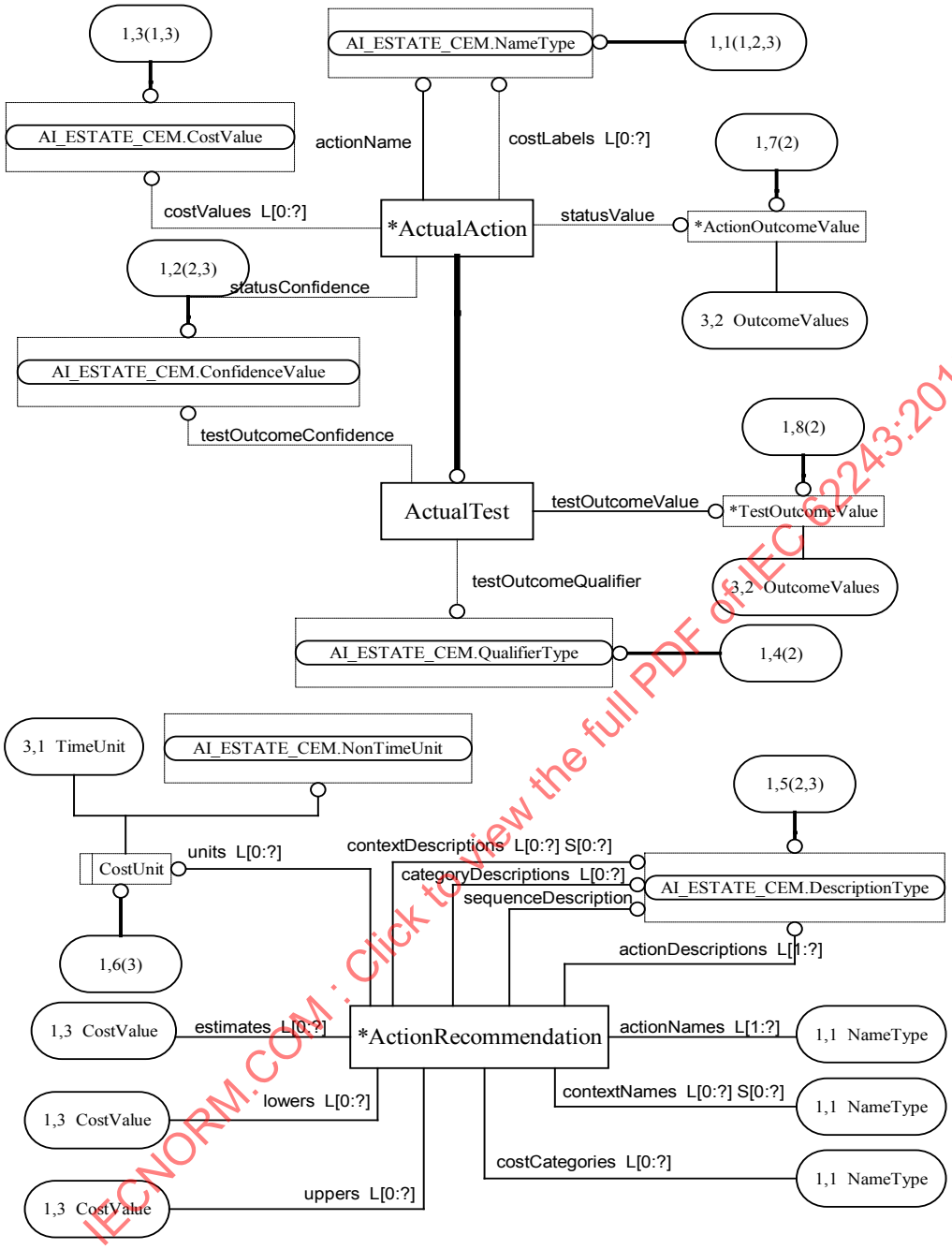


Figure 17—AI_ESTATE_RSM EXPRESS-G diagram 1 of 3

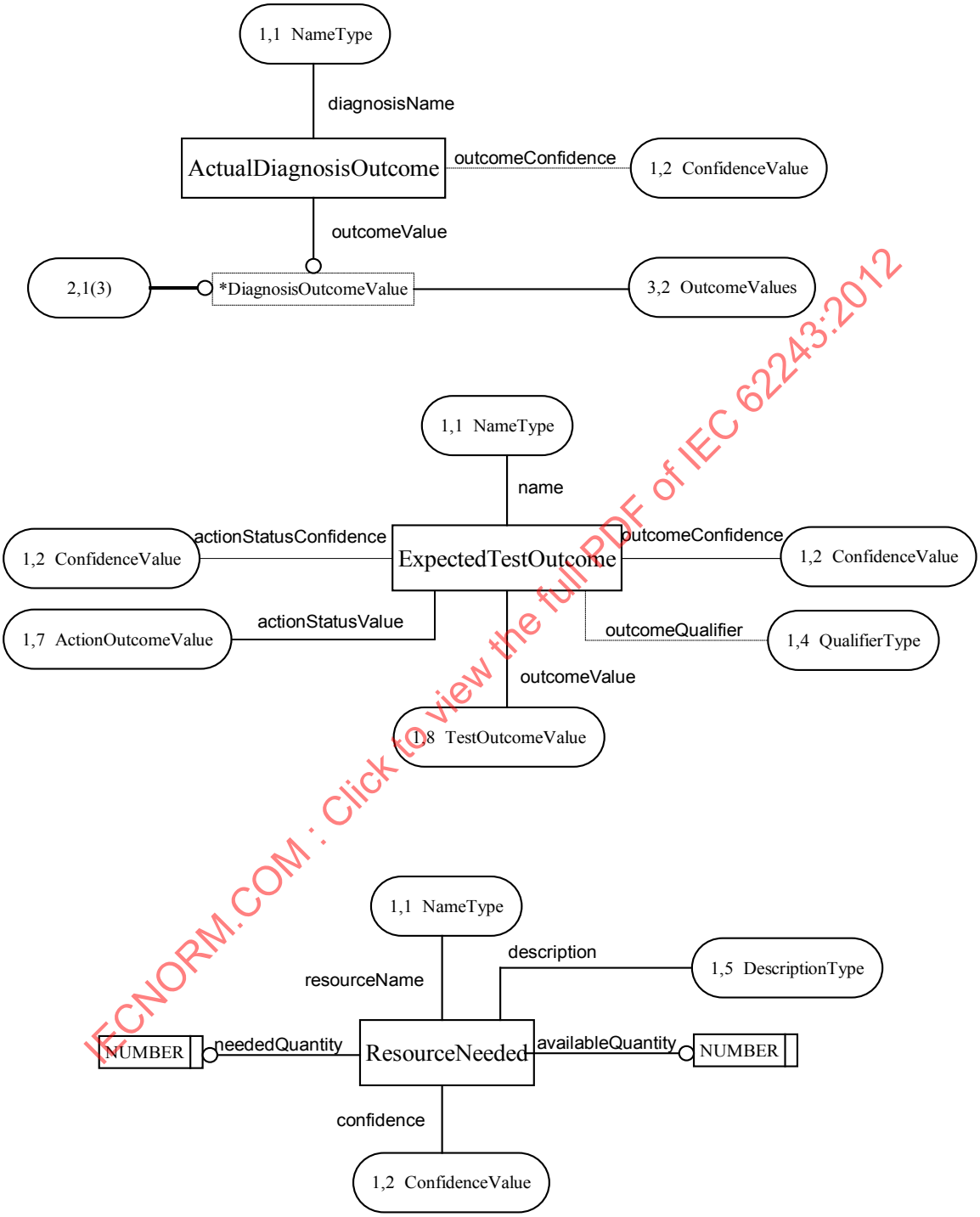


Figure 18—AI_ESTATE_RSM EXPRESS-G diagram 2 of 3

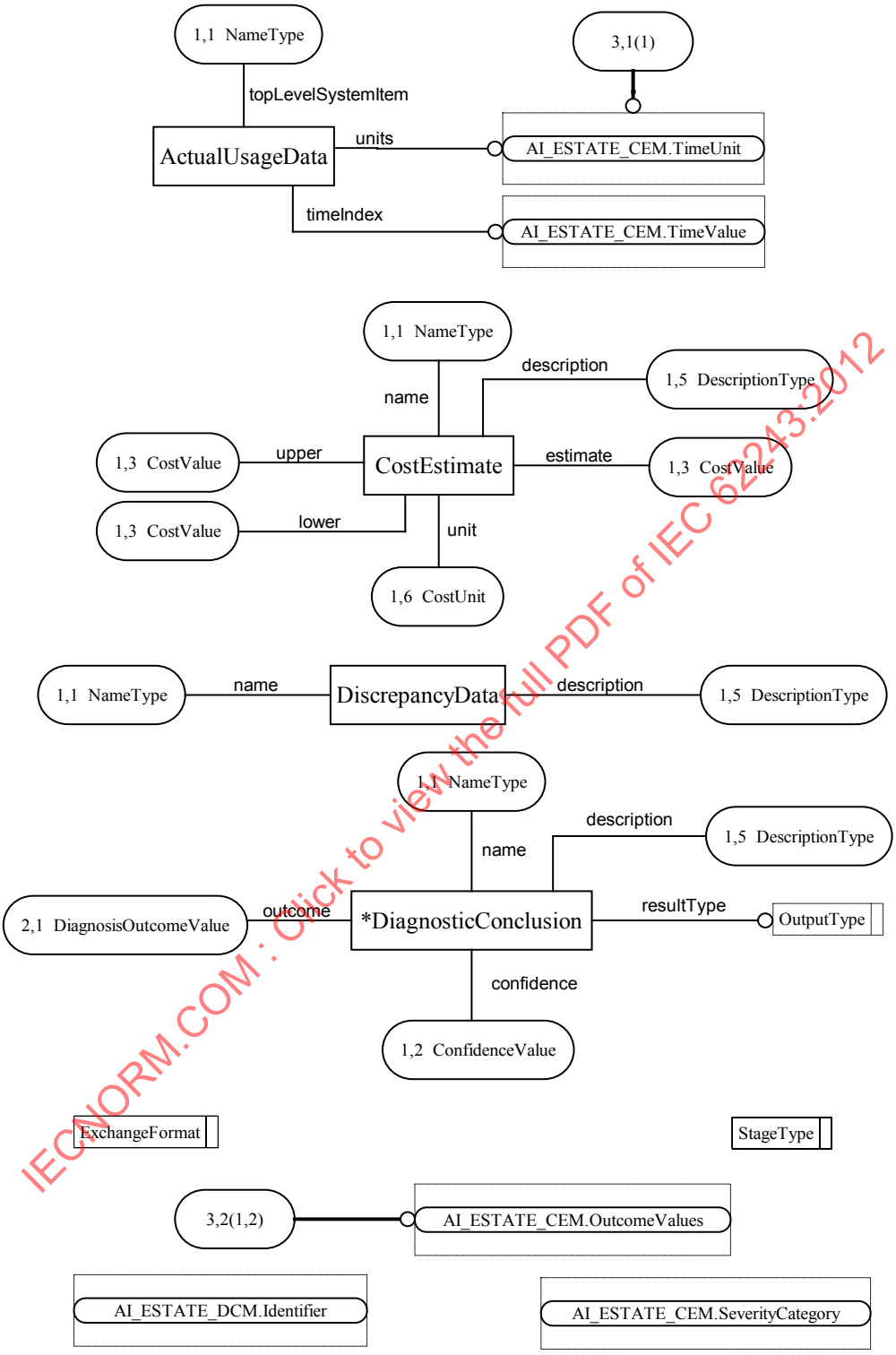


Figure 19—AI_ESTATE_RSM EXPRESS-G diagram 3 of 3

*)

(*

7.4 Required services

This subclause describes the subset of the Reasoner Manipulation Services that are required to claim conformance to Reasoner Manipulation Services.

Some service descriptions in 7.4.1 through 7.4.28 have a subclause named “Effect on DCM” that defines any effects on the DCM instance being generated during the session beyond what is described in the table and rules in 7.1. The full effect on the DCM instance is defined by the table and rules in 7.1 combined with the “Effect on DCM” subclause within each service description.

```
*)

(*
```

7.4.1 applyActions

Service “applyActions” informs the Reasoner of each action that is performed, any outcomes that were observed, and any costs that were incurred.

EXPRESS specification:

```
*)
   PROCEDURE applyActions
      (actions : LIST [1:?] OF ActualAction);
   END_PROCEDURE;
(*
```

Parameter name	Type	Description
actions	LIST [1:?] OF ActualAction	Structure that identifies the actions that were performed in order, and any outcomes, and costs incurred.

Exceptions:

- MISSING_OR_INVALID_ARGUMENT exception shall be raised if actionName is not present or invalid.
- MISSING_OR_INVALID_ARGUMENT exception shall be raised if actionStatusConfidence is present while actionStatusValue is missing.
- MISSING_OR_INVALID_ARGUMENT exception shall be raised if testOutcomeConfidence or testOutcomeQualifier is present while testOutcomeValue is missing.
- MISSING_OR_INVALID_ARGUMENT exception shall be raised if the length of costLabels and costValues are different.
- MISSING_OR_INVALID_ARGUMENT exception shall be raised if costLabels or costValues contains a NULL item.
- MISSING_OR_INVALID_ARGUMENT exception shall be raised if statusValue and testOutcomeValue are both NULL.
- OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— The parameter “actions” contains ActualAction enties and/or subtypes of ActualAction.

Effect on DCM:

— For each ActualAction in the list, and in that order, a corresponding ActiveAction is instantiated and added to the LIST Session.trace[end].actionsPerformed.

— If an ActualAction includes a statusValue, a corresponding ActualOutcome is added to the SET Session.trace[end].Step.outcomesObserved. If the ActualAction includes a statusConfidence, the ActualOutcome will also include an actual confidence with the value given by the statusConfidence.

— If an ActualAction includes a testOutcomeValue, a corresponding ActualOutcome is added to the SET Session.trace[end].Step.outcomesObserved. If the ActualAction includes a testOutcomeConfidence, the ActualOutcome will also include an actual confidence with the value testOutcomeConfidence.

— If an action is performed multiple times during a single Step, only the last outcome is recorded in outcomesObserved.This is because the DCM records outcomesObserved as a SET and has no means of recording the order of observations or cases of many repeated observations with potentially conflicting outcomes.

— If an ActualAction includes costLabels and costValues, then for each item in costLabels, a corresponding ActualCost is instantiated and added to ActiveAction.costIncurred. The value of each ActualCost.actualValue is given by costValues.

7.4.2 applyDiagnosticOutcome

Service “applyDiagnosticOutcome” applies specific values of diagnostic outcomes to the current state of the reasoner. Service “applyDiagnosticOutcome” shall only apply a diagnostic outcome at most one time at a single step.

EXPRESS specification:

```
*)
  PROCEDURE applyDiagnosticOutcome
    (outcomes : LIST [1:?] OF ActualDiagnosisOutcome);
  END_PROCEDURE;
(*
```

Parameter name	Type	Description
outcomes	LIST [1:?] OF ActualDiagnosisOutcome	Structure that identifies the list of diagnostic outcomes that are known at this point of the diagnostic process that have not been inferred by the diagnostic reasoner.

Exceptions:

— MISSING_OR_INVALID_ARGUMENT exception shall be raised if diagnosisName is not present or invalid.

— MISSING_OR_INVALID_ARGUMENT exception shall be raised if statusConfidence is present while statusValue is missing.

— MISSING_OR_INVALID_ARGUMENT exception shall be raised if diagnosisOutcomeConfidence is present while diagnosisOutcomeValue is missing.

— MISSING_OR_INVALID_ARGUMENT exception shall be raised if statusValue and diagnosisOutcomeValue are both NULL.

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— None

Effect on DCM:

— For each ActualDiagnosisOutcome in parameter "outcomes," an ActualOutcome for the associated Diagnosis is added to the set Step.outcomesObserved for the current Step.

7.4.3 backtrack

Service "backtrack" reverts the Session to a previous Step.

EXPRESS specification:

```
*)
  PROCEDURE backtrack
    (numberOfSteps : INTEGER);
  END_PROCEDURE;
(*
```

Parameter name	Type	Description
numberOfSteps	INTEGER	Number of steps to backtrack

Exceptions:

— MISSING_OR_INVALID_ARGUMENT exception shall be raised if numberOfSteps is less than 0 or greater than the length of Session.trace[].

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— Refer to 7.4.27 for notes on backtrack, waypoint, checkpoint, and resume.

— The service reverts to the initial state of the specified step before any services were called.

— numberOfSteps=0 reverts to the current step in its initial state.

— numberOfSteps=1 reverts to the previous step, etc.

— Whereas the DCM records all steps instantiated during a diagnostic session, a backtrack behaves as if steps are popped from a stack. The popped steps, although not removed from Session.trace[], are no longer considered during subsequent backtracks.

Effect on DCM:

— Appends a new Step to Session.trace[] that equals the initial state of the Step that this service reverts to, as specified above in Service details. This becomes the current Step.

— The value of Step.reverted for the new Step points back to the Step that this service reverted to.

7.4.4 closeDiagnosticProcess

Service “closeDiagnosticProcess” terminates the diagnostic session.

EXPRESS specification:

```
*)
  PROCEDURE closeDiagnosticProcess
    ;
  END_PROCEDURE;
(*
```

Exceptions:

None

Service details:

— This service does not automatically save the Session.

— The session cannot be resumed after calling this service.

Effect on DCM:

— The Session is no longer available to the reasoner.

7.4.5 describeReasoner

Service “describeReasoner” provides a description of the reasoner.

EXPRESS specification:

```
*)
  FUNCTION describeReasoner
    : STRING;
  END_FUNCTION;
(*
```

Returns:

String containing implementation-specific reasoner configuration and identification information.

Exceptions:

None

Service details:

— The format and content of the return string is implementation dependent.

7.4.6 getDiagnosticResults

Service "getDiagnosticResults" returns a list of inferred diagnoses. The input parameters define the subset of diagnostic conclusions to return.

EXPRESS specification:

```
*)
  FUNCTION getDiagnosticResults
     (outcomeOfInterest   : DiagnosisOutcomeValue;
     minConfidence        : ConfidenceValue;
     maxNumber            : INTEGER;
     levelsOfInterest     : SET [0:?] OF NameType;
     outputOfInterest     : OutputType): LIST OF DiagnosticConclusion;
  END_FUNCTION;
(*
```

Parameter name	Type	Description
outcomeOfInterest	DiagnosisOutcomeValue	Directs the service to return only outcomes of this value
minConfidence	ConfidenceValue	The minimum outcome confidence to return
maxNumber	INTEGER	Maximum number of conclusions to return
levelsOfInterest	SET [0:?] OF NameType	Defines a set of Levels used for filtering the conclusions according to their associated Level
outputOfInterest	OutputType	Defines the type of diagnosis that will be returned

Returns:

List of DiagnosisConclusion. If confidence information is available, the list is sorted highest to lowest based on the inferred confidences. Otherwise, sorting is unspecified.

Exceptions:

— MISSING_OR_INVALID_ARGUMENT exception shall be raised if outcomeOfInterest is not a valid diagnosis outcome for any diagnoses.

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— The service supports each of the types of diagnostic conclusion supported by the entity DiagnosticConclusion.

— maxNumber<=0 directs the service to not limit the size of the return list. maxNumber>0 directs the service to return no more than maxNumber entries.

— minConfidence=0 directs the service to not filter the conclusions by confidence. minConfidence>0 directs the service to only return conclusions with confidence greater than or equal to minConfidence.

— The service ignores the minConfidence parameter if confidence information is not included in the active diagnostic model. In this case, conclusions are not filtered by confidence.

— The parameter levelsOfInterest directs the service to return only conclusions associated with Level entities with the names listed within levelsOfInterest. If Level information is not included in the active diagnostic model, then the service ignores this parameter and conclusions are not filtered by Level. An empty set levelOfInterest parameter directs the service to return conclusions associated with any Level. A null string in the set directs the service to include items not associated with a Level.

7.4.7 initializeDiagnosticProcess

Service “initializeDiagnosticProcess” begins a diagnostic session for the specified system under test.

EXPRESS specification:

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

Parameter name	Type	Description
itemID	Identifier	The unique identifier for the system being diagnosed, such as a serial number
systemItemName	NameType	Name attribute of the actual system item under test

Returns:

Returns the name of the session.

Exceptions:

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Effect on DCM:

— Creates the ActualSystemItem, and sets ActualSystemItem.diagnosedItemID to the value of the itemID parameter.

— The attribute ActualSystemItem.itemUnderTest will be set toDiagnosticModel.systemUnderTest once a DiagnosticModel is loaded.

— Creates the Session and sets Session.diagnosedItem to the ActualSystemItem.

— Assigns an implementation-specific unique value to Session.name.

— Sets Session.timeInitiated to the current date and time.

7.4.8 loadDiagnosticModel

Service “loadDiagnosticModel” loads a DiagnosticModel and makes it available for reasoning.

EXPRESS specification:

```
*)
   PROCEDURE loadDiagnosticModel
      (name : NameType);
   END_PROCEDURE;
(*
```

Parameter name	Type	Description
name	NameType	Name of the diagnostic model

Exceptions:

— NONEXISTENT_DATA_ELEMENT_REQUESTED exception shall be raised if there is no DiagnosticModel whose name attribute matches the name specified.

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

— MISSING_OR_INVALID_ARGUMENT exception shall be raised if the DiagnosticModel.systemUnderTest.name does not match the systemItemName provided to initializeDiagnosticProcess service.

Service details:

— The input parameter name must match the name attribute of a DiagnosticModel.

— The application exposing the service handles the details of how and where the DiagnosticModel is stored and accessed.

Effect on DCM:

— Sets Session.modelForDiagnosis to equal the DiagnosticModel that was loaded.

— Sets ActualSystemItem.itemUnderTest to equal DiagnosticModel.systemUnderTest.

— The data in the loaded model become available for use in the DCM.

— This standard specifies that there be exactly one DiagnosticModel in Session.modelForDiagnosis. As such, calling this service will replace any previously loaded model data with the newly loaded model, and only the data in the most recently loaded model are available for use in the DCM.

7.4.9 loadDiagnosticModelFromLocation

Service “loadDiagnosticModelFromLocation” loads a DiagnosticModel from a client-specified location.

EXPRESS specification:

```
*)
   FUNCTION loadDiagnosticModelFromLocation
      (location    : URI;
      modelFormat  : ExchangeFormat) : SET OF NameType;
   END_FUNCTION;
(*
```

Parameter name	Type	Description
location	URI	A URI fully specifying the location of the model
format	ExchangeFormat	Format of the saved Model: P21, P28, or NATIVE

Returns:

Returns the name attribute of the DiagnosticModel that is loaded from the location.

Exceptions:

— NONEXISTENT_DATA_ELEMENT_REQUESTED exception shall be raised if the DiagnosticModel does not exist at the location specified.

— MISSING_OR_INVALID_ARGUMENT exception shall be raised if the format specified is invalid or is not supported by this reasoner.

— INVALID_MODEL_SCHEMA exception shall be raised if the model type is not supported by the reasoner (e.g., a Dmatrix).

— INVALID_MODEL_SCHEMA exception shall be raised if the data is invalid.

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence

— MISSING_OR_INVALID_ARGUMENT exception shall be raised if the DiagnosticModel.systemUnderTest.name does not match the systemItemName provided to initializeDiagnosticProcess service.

Service details:

— This service shall support at least one of the formats “ISO 10303-21” or “ISO 10303-28” as defined in 5.1 and may also support “Native.”

— If the service supports loading a standard AI-ESTATE diagnostic model type (e.g., Dmatrix), then it shall be capable of loading extended models of that type as well, including extensions defined by other parties. For extensions defined by other parties, the reasoner shall at least consume the portions of the model defined by standard AI-ESTATE.

— The service shall fully validate any standard AI-ESTATE exchange model that it loads. For an extended model, the service must at least validate the portions defined by standard AI-ESTATE.

— The BNM, DIM, DLM, and FTM specify there be exactly one DiagnosticModel per exchange file. This service returns the DiagnosticModel name in a SET allowing for possible expansion to multiple DiagnosticModels per file in the future.

Effect on DCM:

— Sets Session.modelForDiagnosis to equal the DiagnosticModel that was loaded.

— Sets ActualSystemItem.itemUnderTest to equal DiagnosticModel.systemUnderTest.

— The data in the loaded model become available for use in the DCM.

— This standard specifies that there be exactly one DiagnosticModel in Session.modelForDiagnosis. As such, calling this service will replace any previously loaded model data with the newly loaded model, and only the data in the most recently loaded model are available for use in the DCM.

7.4.10 loadHistoryFromLocation

Service “loadHistoryFromLocation” loads a DCM instance from a client-specified location and includes it in the history for the current system under test.

EXPRESS specification:

```
*)
   FUNCTION loadHistoryFromLocation
      (location    : URI;
       modelFormat  : ExchangeFormat) : SET OF NameType;
   END_FUNCTION;
(*
```

Parameter name	Type	Description
location	URI	A URI fully specifying the location of the model
format	ExchangeFormat	Format of the saved Model: P21, P28, or NATIVE

Returns:

Returns the name attribute of each session that was loaded from the location.

Exceptions:

— NONEXISTENT_DATA_ELEMENT_REQUESTED exception shall be raised if the DCM instance does not exist at the location specified.

— MISSING_OR_INVALID_ARGUMENT exception shall be raised if the format specified is invalid, or not supported by this reasoner.

— INVALID_MODEL_SCHEMA exception shall be raised if the data is invalid.

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

— No error occurs if the same Session is loaded more than once during a session.

Service details:

- This service shall support at least one of the formats “ISO 10303-21” or “ISO 10303-28” as defined in 5.1 and may also support “Native.”
- In this standard. the DCM allows exactly one Session per exchange file. As such, this service will return a set constisting of one session name. This service returns a set allowing for possible expansion to multiple Sessions per file in future versions of AI-ESTATE.
- The service shall support loading a standard AI-ESTATE DCM instance and shall be capable of loading extended DCM instances as well, including extensions defined by other parties. For extensions defined by other parties, the reasoner shall at least consume the portions of the model defined by standard AI-ESTATE.
- The service shall fully validate any standard AI-ESTATE exchange instance that it loads. For an extended instance, the service must at least validate the portions defined by standard AI-ESTATE.

Effect on DCM:

- Adds the URI parameter to the SET ActualSystemItem.history if it is not already present.

7.4.11 pauseDiagnosticProcess

Service “pauseDiagnosticProcess” temporarily stops the diagnostic session so that it can be resumed at a later time.

EXPRESS specification:

```
*)
  PROCEDURE pauseDiagnosticProcess;
  END_PROCEDURE;
(*
```

Exceptions:

- OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

- Refer to 7.4.28 for notes on backtrack, waypoint, checkpoint, and resume.
- This service does not automatically save the Session. Any data that an application may save as the underlying mechanism for this service is implementation specific.
- After pausing the Session, the session can be resumed using the service resumeDiagnosticSession.

Effect on DCM:

- The Session is not available to the reasoner until it is resumed.

7.4.12 recommendActions

Service “recommendActions” returns a sorted list of recommended actions to perform.

EXPRESS specification:

```
*)
  FUNCTION recommendActions
     (maxNumber       : INTEGER;
      levelsOfInterest : SET OF NameType) : LIST OF ActionRecommendation;
  END_FUNCTION;
(*
```

Parameter name	Type	Description
maxNumber	INTEGER	Maximum number of recommendations to return
levelsOfInterest	SET [1:?] of NameType	Defines a set of Levels used for filtering the recommended actions according to their associated Level

Returns:

Returns a list of ActionRecommendations, sorted best to worst. If optimization information is available, sorting is based on that information otherwise sorting is unspecified. Each ActionOutput is an alternative path forward from which the client may choose.

Exceptions:

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— maxNumber<=0 directs the service to not limit the size of the return list. maxNumber>0 directs the service to return no more than maxNumber entries.

— The parameter levelsOfInterest directs the service to return only Actions associated with Level entities with the names listed within levelsOfInterest. A null string in the set directs the service to include Actions not associated with a Level. If Level information is not included in the active diagnostic model, then the service ignores this parameter and recommended actions are not filtered by Level. An empty set levelOfInterest parameter directs the service to return Actions associated with any Level.

7.4.13 requestResourcesNeeded

Returns the resources that are necessary to complete a given a set of Actions.

EXPRESS specification:

```
*)
   FUNCTION requestResourcesNeeded
      (actionNames : SET [1:?] OF NameType) : SET [0:?] OF
         ResourceNeeded;
   END_FUNCTION;
(*
```

Parameter name	Type	Description
actionNames	SET [1:?] OF NameType	A set of action names

Returns:

A set of ResourceNeeded entities that describes the resources needed for a specified set of actions.

Exceptions:

— NONEXISTENT_DATA_ELEMENT_REQUESTED exception shall be raised if name in "actions" does not match an Action.name in an available diagnostic model.

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— The names in "actionNames" refer to Actions in the active diagnostic model by their Action.name attributes.

— The returned set of ResourcesNeeded will be empty if no resources are required or if the application or diagnostic models do not support Resources.

7.4.14 restoreCheckpoint

Restores a Session to a state that was previously saved by calling setCheckpoint.

EXPRESS specification:

```
*)
   FUNCTION restoreCheckpoint
      (checkpointName : NameType) : NameType;
   END_FUNCTION;
(*
```

Parameter name	Type	Description
checkpointName	NameType	Name of the checkpoint to retrieve

Returns:

Name of the session.

Exceptions:

— NONEXISTENT_DATA_ELEMENT_REQUESTED exception shall be raised if an attempt is made to restore a checkpoint that does not exist.

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— Refer to 7.4.28 for notes on backtrack, waypoint, checkpoint, and resume.

— The parameter checkpointName shall equal the name assigned to the checkpoint in the earlier call to setCheckpoint.

— The application exposing the service handles the details of how and where the underlying session data is managed to support this service.

Effect on DCM:

— Reverts the Session as it existed when the checkpoint of the given name was set.

7.4.15 restoreWaypoint

Reverts the Session to a previous Step.

EXPRESS specification:

```
*)
   PROCEDURE restoreWaypoint
      (waypointName : NameType);
   END_PROCEDURE;
(*
```

Parameter name	Type	Description
waypointName	NameType	Name of the waypoint to revert to

Exceptions:

— NONEXISTENT_DATA_ELEMENT_REQUESTED exception shall be raised if an attempt is made to restore a waypoint that does not exist.

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— Refer to 7.4.28 for notes on backtrack, waypoint, checkpoint, and resume.

— Reverts back to the initial state of the Step which was previously designated as the waypoint with this waypointName via a call to setWaypoint.

Effect on DCM:

— Appends a new Step to Session.trace[] that equals the initial state of the Step that this service reverts to, as specified above in Service details. This becomes the current Step.

— The value of Step.reverted for the new Step points back to the Step that this service reverted to.

7.4.16 resumeDiagnosticProcess

Service "resumeDiagnosticProcess" restarts a diagnostic session that was stopped using "pauseDiagnosticProcess."

EXPRESS specification:

```
*)
   PROCEDURE resumeDiagnosticProcess
      (sessionName : NameType);
   END_PROCEDURE;
(*
```

Parameter name	Type	Description
sessionName	NameType	Name of the session to resume

Exceptions:

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— Refer to 7.4.28 for notes on backtrack, waypoint, checkpoint, and resume.

Effect on DCM:

— Resumes the session.

— The application exposing the service handles the details of how and where the underlying session data is managed to support this service.

7.4.17 saveSession

Saves a Session.

EXPRESS specification:

```
*)
   PROCEDURE saveSession
      (modelFormat : ExchangeFormat);
   END_PROCEDURE;
(*
```

Parameter name	Type	Description
format	ExchangeFormat	Format of the saved Model: P21, P28, or NATIVE

Exceptions:

- MISSING_OR_INVALID_ARGUMENT exception shall be raised if the format specified is invalid.
- OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

- If a saved Session with the same Session.name already exists, it will be overwritten.
- The application exposing the service handles the details of how and where the Session is stored.
- A Session entity saved by this service is usable as history for a future Session.
- The application exposing the service handles the details of how and where the underlying session data is managed to support this service.

7.4.18 saveSessionToLocation

Saves a Session to a specified location.

EXPRESS specification:

```
*)
  PROCEDURE saveSessionToLocation
    (location    : URI;
     modelFormat : ExchangeFormat);
  END_PROCEDURE;
(*
```

Parameter name	Type	Description
location	URI	A URI fully specifying the location where the session should be saved
format	ExchangeFormat	Format of the saved Model: P21, P28, or NATIVE

Exceptions:

- NONEXISTENT_DATA_ELEMENT_REQUESTED exception shall be raised if the application could not write to the specified location.
- MISSING_OR_INVALID_ARGUMENT exception shall be raised if the format specified is invalid.
- OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— If a Session at location already exists, it will be overwritten.

— A Session entity saved by this service is usable as history for a future Session.

7.4.19 setActiveModel

Service "setActiveModels" makes the specified DiagnosticModel active for reasoning from this point forward.

EXPRESS specification:

```
*)
   PROCEDURE setActiveModel
      (diagnosticModelNames : SET [1:1] OF NameType);
   END_PROCEDURE;
(*
```

Parameter name	Type	Description
diagnosticModelNames	SET [1:1] OF NameType	Name attribute of the DiagnosticModel to make the active model

Exceptions:

— NONEXISTENT_DATA_ELEMENT_REQUESTED exception shall be raised if the named model is not already loaded.

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— The DiagnosticModel specified by diagnosticModelNames must already be loaded and available.

— The specified DiagnosticModel and its associated data become the knowledge base used by the reasoner for the remainder of the session.

— This standard specifies exactly one DiagnosticModel per session. The parameter diagnosticModelNames is a set to support possible future expansion.

Effect on DCM:

— Instantiate a new Step with attributes set to their intial state, and make that Step the only Step in Session.trace.

— The data associated with the DiagnosticModel become available for use in the DCM.

7.4.20 setAvailableResources

Tells the reasoner what resources are available.

EXPRESS specification:

```
*)
  PROCEDURE setAvailableResources
    (resourceNames  : LIST [1:?] OF NameType;
    Quantities      : LIST [1:?] OF NUMBER);
  END_PROCEDURE;
(*
```

Parameter name	Type	Description
resourceNames	LIST [1:?] OF NameType	Names of the available resources
quantities	LIST [1:?] OF NUMBER	Corresponding quantity for each resource

Exceptions:

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— There must be an active diagnostic model to call this procedure.

— This procedure may be called during any step in the session.

— The value of the of each resourceNames must match a Resource.name in an available diagnostic model.

— All Resources are assumed available at the start of a session.

— Resource availability information remains in effect until the end of the session, or until it is asserted by a call to this procedure.

— The length of quantities must equal the length of resourceNames.

— A quantity=0 indicates the resource is not available. A quantity>0 indicates the quantity of the resource available. A quantity<0 indicates unknown availability.

Effect on DCM:

— For resources that are designated as available, adds the Resource of that name in the available diagnostic model to the set Step.availableResouces in the current Step. The numerical quantity of the available resource is not captured in the DCM.

— For resources that are designated as not available or unknown, removes the Resource of that name in the available diagnostic model from the set Step.availableResouces in the current Step.

7.4.21 setCheckpoint

Saves the current state of the diagnostic session.

EXPRESS specification:

```
*)
```

```
   FUNCTION setCheckpoint
      (checkpointName : NameType) : NameType;
   END_FUNCTION;
(*
```

Parameter name	Type	Description
checkpointName	NameType	Checkpoint name

Returns:

Checkpoint name.

Service details:

— Refer to 7.4.28 for notes on backtrack, waypoint, checkpoint, and resume.

— If checkpointName is not specified, the Reasoner will create one, and that name will be returned. If a checkpointName is specified, that name will still be returned as an echo by the service.

— If a checkpoint with checkpointName already exists, it will be overwritten.

— The format of the Session saved is implementation dependent.

— The application exposing the service handles the details of how and where the underlying session data is managed to support this service.

7.4.22 setContext

Tells the reasoner the current context state.

EXPRESS specification:

```
*)
   PROCEDURE setContext
      (contextName : NameType);
   END_PROCEDURE;
(*
```

Parameter name	Type	Description
contextName	NameType	Name of the ContextState in the static model that is instantiated

Exceptions:

— OPERATION_OUT_OF_SEQUENCE exception shall be raised if the service is called out of sequence.

Service details:

— There must be an active diagnostic model to call this procedure.

— This procedure may be called during any step in the session.