INTERNATIONAL STANDARD

ISO/IEC/ IEEE 31320-1

First edition
2012-09-15

Information technology — Modeling Languages —

Part 1: Syntax and Semantics for IDEF0

Technologies de l'information — Langages de modélisation —

Partie 1: Syntaxe et sémantique pour IDEF0

Reference number
ISO/IEC/IEEE 31320-1:2012(E)

ISO copyright office
Case postale 56
CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
E-mail inmail@iec.ch
Web www.iec.ch

Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York
NY 10016-5997, USA
E-mail stds.ipr@ieee.org
Web www.ieee.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

The main task of ISO/IEC JTC 1 is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is called to the possibility that implementation of this standard may require the 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. ISO/IEEE is not responsible for identifying essential patents or patent claims for which a license may be required, for conducting inquiries into the legal validity or scope of patents or patent claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance or a Patent Statement and Licensing Declaration Form, 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 ISO or the IEEE Standards Association.

ISO/IEC/IEEE 31320-1 was prepared by the Software & Systems Engineering Standards Committee of the IEEE Computer Society (as IEEE 1320.1-1998). It was adopted by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*, in parallel with its approval by the ISO/IEC national bodies, under the “fast-track procedure” defined in the Partner Standards Development Organization cooperation agreement between ISO and IEEE. IEEE is responsible for the maintenance of this document with participation and input from ISO/IEC national bodies.

ISO/IEC/IEEE 31320 consists of the following parts:

— ISO/IEC/IEEE 31320-1, *Information technology — Modeling Languages — Part 1: Syntax and Semantics for IDEF0*

— ISO/IEC/IEEE 31320-2, *Information technology — Modeling Languages — Part 2: Syntax and Semantics for IDEF1X$_{97}$ (IDEF$_{object}$)*

(blank page)

IEEE Std 1320.1-1998(R2004)

IEEE Standard for Functional Modeling Language—Syntax and Semantics for IDEF0

Sponsor

Software Engineering Standards Committee of the IEEE Computer Society

Approved 25 June 1998

IEEE-SA Standards Board

Abstract: IDEF0 function modeling is designed to represent the decisions, actions, and activities of an existing or prospective organization or system. IDEF0 graphics and accom-panying texts are presented in an organized and systematic way to gain understanding, support analysis, provide logic for potential changes, specify requirements, and support system-level design and integration activities. IDEF0 may be used to model a wide variety of systems, composed of people, machines, materials, computers, and information of all varieties and structured by the relationships among them, both automated and nonautomated. For new systems, IDEF0 may be used first to define requirements and to specify functions to be carried out by the future system. As the basis of this architecture, IDEF0 may then be used to design an implementation that meets these requirements and performs these functions. For existing systems, IDEF0 can be used to analyze the functions that the system performs and to record the means by which these are done.

Keywords: enterprise, functional modeling language, IDEF0, language, modeling language

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

ISBN 0-7381-0340-3

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

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

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

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

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

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

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

Introduction

(This introduction is not part of IEEE Std 1320.1-1998, IEEE Standard for Functional Modeling Language—Syntax and Semantics for IDEF0.)

This standard formally documents the Integration Definition 0 (IDEF0) language for function models in two parts. The body of this standard describes the syntax and semantics of the IDEF0 language that are required to draw the physical diagrams of a *specific* IDEF0 model. Annex B describes the syntax and semantics of the IDEF0 language as an *abstract* formal structure and, therefore, provides the foundation for the specifics found in the body of the standard. The diagrams discussed by the standard are real instantiations in a concrete model of the mathematical formalisms of the IDEF0 Language Abstract Formalization (the “language formalization”).

Background

During the 1970s, the US Air Force Program for Integrated Computer Aided Manufacturing (ICAM) sought to increase manufacturing productivity through systematic application of computer technology. The ICAM program identified the need for better analysis and communication techniques for people involved in improving manufacturing productivity.

As a result, the ICAM program developed a series of modeling methodologies known as the ICAM Definition (IDEF) methods, which include

a) IDEF0, used to produce a function model. A function model is a structured representation of the functions within a system or subject area.

b) IDEF1, used to produce an information model. An information model represents the structure and semantics of information within a system or subject area.

c) IDEF2, used to produce a dynamics model. A dynamics model represents the behavior of a system or subject area as it varies over time.

IDEF0 was derived from a well-established graphical modeling method known as the Structured Analysis and Design Technique (SADT). IDEF0 was developed by the originators of SADT, notably Douglas T. Ross, under the ICAM program. In 1983, the US Air Force Integrated Information Support System program enhanced the IDEF1 information modeling technique to develop IDEF1 extended (IDEF1X), a semantic data modeling method.

Continued IDEF method developments followed to address needs for additional analytic methods. These follow-on developments have been directed toward providing a mutually supportive family of methods that are applicable to a broad range of enterprise improvement and integration strategies (e.g., concurrent engineering, total quality management, business reengineering). Reflecting this general applicability, the IDEF acronym has been recast to refer to an integrated family of Integration Definition methods. Currently, IDEF0 and IDEF1X techniques are widely used in the government, industrial, and commercial sectors, supporting modeling efforts for a wide range of enterprises and application domains. IDEF0 has been widely adopted as the function modeling method of choice in large number of both military and nonmilitary organizations in both North America and Europe.

In 1991, the National Institute of Standards and Technology (NIST) received support from the US Department of Defense, Office of Corporate Information Management (DoD/CIM), to develop Federal Information Processing Standards (FIPS) for modeling techniques for use by the federal government. One

product of this effort was FIPS PUB 183, *Integration Definition for Function Modeling (IDEF0)* [B2].[a] This FIPS document was based on the IDEF specification manuals published by the US Air Force in the early 1980s. At the same time, NIST also published FIPS PUB 184, *Integration Definition for Information Modeling (IDEF1X)* [B3], to support data modeling for the federal government.

In 1993, the Institute of Electrical and Electronics Engineers (IEEE) Computer Society initiated a project to establish IDEF standards across both industry and government within the standards framework of the American National Standards Institute (ANSI). IEEE Std 1320.1-1998 for IDEF0 function models, based on FIPS PUB 183 [B2], is a result of that effort.

This standard is explicitly oriented to the presentation of an IDEF0 model on paper pages; development of an IDEF0 standard for other presentation media is deliberately not addressed by this document. However, integrated into this standard is a mathematically founded formalization of an IDEF0 model. This formalization allows users to separate what they conceive in conceptual space, that is, the model itself, from their presentation of that model and from their presentation media. In earlier work, the IDEF0 diagram was not considered as something that should or could be distinguished from the paper page that presents that diagram. However, the current formalization allows (indeed, forces) users to separate the abstract structure of an IDEF0 model from the physical structure of the presentation of that model using sheets of paper. An important conceptual and notational difference between this work and earlier work is the clear distinction between an IDEF0 graphic diagram and the medium, e.g., a page of paper, that is used to present that diagram.

New terminology has been presented to ensure that this distinction can be easily maintained, and a more robust categorization of both the components of an IDEF0 model and of an IDEF0 diagram has been introduced. This terminology ensures that the usage presented in the body of this standard is consistent with the formalization presented in Annex B. In addition, these changes to the IDEF0 vocabulary will facilitate the development of IDEF0 presentations in digital or other media.

The IDEF0 approach

IDEF0 includes both a modeling language and a comprehensive methodology for developing models. This standard addresses only the syntax and semantics of the modeling language itself.

In addition to definition of the IDEF0 language, the IDEF0 methodology also prescribes procedures and techniques for developing and interpreting models, including ones for data gathering, diagram construction, review cycles, and documentation.

IDEF0 function modeling is designed to represent the decisions, actions, and activities of an existing or prospective organization or system. For all its apparent simplicity, the method is surprisingly powerful and effective. Like most modeling methods, the primary component of IDEF0 is a graphical language whose constructs are intended to convey information of a certain sort. IDEF0 graphics and accompanying texts are presented in an organized and systematic way to gain understanding, support analysis, provide logic for potential changes, specify requirements, and support systems-level design and integration activities. IDEF0 may be used to model a wide variety of systems, composed of people, machines, materials, computers, and information of all varieties and structured by the relationships among them, both automated and nonautomated. For new systems, IDEF0 may be used first to define requirements and to specify functions to be carried out by the future system. As the basis of this architecture, IDEF0 may then be used to design an implementation that meets these requirements and performs these functions. For

[a]The numbers in brackets correspond to those of the bibliography in Annex A.

existing systems, IDEF0 can be used to analyze the functions that the system performs and to record the means by which these are done.

The result of applying IDEF0 to a system is a model that consists of a hierarchical series of diagrams, with accompanying explanatory text, illuminating graphical, and defining glossary pages that are cross-referenced to these diagrams. The two primary modeling components of a diagram are functions (represented by named boxes) and the physical and data objects that interrelate those functions (represented by labeled arrows).

As a function modeling language, IDEF0 has these characteristics:

a) It is comprehensive and expressive, capable of graphically representing a wide variety of business, manufacturing, and other types of enterprise operations to any level of detail.

b) It is a coherent and simple language, allowing rigorous and precise expression and promoting consistency of usage and interpretation.

c) It enhances communication among analysts, architects, developers, managers, and users through its ease of learning and its emphasis on hierarchical exposition of detail.

d) It is well-tested and proven through many years of use by the USAir Force and other government agencies and by private industry.

e) It can be generated by a variety of computer-based tools; several commercial products specifically support development and analysis of IDEF0 diagrams and models.

As a system engineering technique, IDEF0 may be used for performing and managing needs analysis, benefits analysis, requirements definition, functional analysis, systems design, maintenance, and baselines for continuous improvement. IDEF0 models provide a “blueprint” of functions and their interfaces that must be captured and understood in order to make systems engineering decisions that are logical, affordable, integratable, and achievable. When used in a systematic way, IDEF0 provides a systems engineering approach to

a) Performing systems analysis and design at all levels, including the entire enterprise, a system, or a subject area;

b) Producing reference documentation concurrent with development to serve as a basis for integrating new systems or improving existing systems;

c) Allowing collaborative team consensus to be achieved by shared understanding;

d) Managing large and complex projects using qualitative measures of progress; and

e) Providing a reference architecture for enterprise analysis, information engineering, and resource management.

Typographic conventions

A word that has a special meaning for IDEF0 is italicized the first time that it is used in its specific sense for IDEF0. There will be an entry in Clause 2 for each italicized word.

Figure conventions

The figures in this document have been prepared using the IDEF standard diagram form (SDF). The SDF and its use are documented by both FIPS PUB 183 [B2] and FIPS PUB 184 [B3]. FIPS PUB 183 [B2] does not standardize the SDF; instead, the SDF is covered in an informative annex. In keeping with this precedent, this standard also refrains from addressing the SDF as an element of IDEF0 language standardization.

The IDEF SDF is designed for use and presentation on conventional 8½-by-11-inch paper sheets. Therefore, to preserve this conventional size, the figures for this document have been collected at the end of the document rather than being interspersed with the text.

Contents

IEEE Standard for Functional Modeling Language— Syntax and Semantics for IDEF0

1. Overview

This standard provides requirements for the construction of semantically and syntactically correct Integration Definition 0 (IDEF0) models and diagrams. A model or diagram conforms to this standard if it adheres to all mandatory provisions (marked by the verbs “shall” or “is”) of this standard.

Annex C of this document collects the figures that are referenced in the body of the standard and in Annex B.

1.1 Scope

The body of the standard describes the modeling language (syntax and semantics) that supports the IDEF0 method for developing graphical representations of a system or subject area. The clauses that follow govern the physical construction of IDEF0 models that represent functions, functional relationships, and the physical and data objects required by those relationships.

This part of the document is divided into 10 clauses. Clause 1 provides an overview of this part of the standard. Clause 2 defines key terms. Clause 3 discusses the concept of an IDEF0 model. Clause 4 defines the syntax of the IDEF0 language. Clause 5 defines the semantics of the language. Clause 6 describes the different types of IDEF0 diagrams. Clause 7 presents the different types of IDEF0 model pages. Clause 8 provides details on the various features of an IDEF0 diagram. Clause 9 defines IDEF0 reference expressions. Finally, Clause 10 defines IDEF0 diagram feature references.

Documentation of best commercial practices and guides to recommended usage are beyond the scope of this document.

1.2 Purpose

The objectives of this part of the standard are to prescribe the construction and components of an IDEF0 model and to define the correct syntax and semantics for construction of an IDEF0 diagram within an IDEF0 model.

2. Definitions, acronyms, and abbreviations

This clause contains definitions of important IDEF0 terms used in this standard. Terms used in a definition that are defined elsewhere in this clause are italicized.

2.1 Definitions

2.1.1 A-0 context diagram: The only *context diagram* that is a required for a valid *IDEF0 model*, the *A-0* diagram contains one *box*, which represents the top-level *function* being modeled, the *inputs*, *controls*, *outputs*, and *mechanisms* attached to this *box*, the full *model name*, the *model name abbreviation*, the model's *purpose statement*, and the model's *viewpoint statement*.

2.1.2 A-0 diagram: See: *A-0 context diagram*.

2.1.3 activation: One occurrence of a *function*'s transformation of some subset of its *inputs* into some subset of its *outputs*.

2.1.4 activation constraint: A *function*'s requirement for the presence of a nonempty *object set* in a particular *arrow role* as a precondition for some *activation* of the *function*.

2.1.5 activity: See: *function*.

2.1.6 ancestral box: A *box* related to a specific *diagram* by a *hierarchically consecutive* sequence of one or more parent/child relationships.

2.1.7 ancestral diagram: A *diagram* that contains an *ancestral box*.

2.1.8 arrow: A directed line, composed of one or more connected *arrow segments* in a single *diagram* from a single source (*box* or *diagram boundary*) to a single use (*box* or *diagram boundary*). See also: *arrow segment*, *boundary arrow*, *internal arrow*.

2.1.9 arrow label: A noun or noun phrase associated with an *arrow segment* to signify the *arrow meaning* of the *arrow segment*. Specifically, an arrow label identifies the *object type set* that is represented by an *arrow segment*.

2.1.10 arrow meaning: The *object types* (e.g., a physical thing, a data element) of an *object type set*, regardless of how these *object types* may be collected, aggregated, grouped, bundled, or otherwise joined within the *object type set*.

2.1.11 arrow reference: See: *ICOM code*.

2.1.12 arrow role: The relationship between an *object type set* represented by an *arrow segment* and the *activity* represented by the *box* to which the *arrow segment* is attached. There are four arrow roles: *input*, *control*, *output*, and *mechanism*.

2.1.13 arrow segment: A directed line that originates at a *box* side, arrow *junction* (*branch* or *join*), or *diagram boundary* and terminates at the next *box* side, arrow *junction* (*branch* or *join*), or *diagram boundary* that occurs in the path of the line.

2.1.14 boundary arrow: An *arrow* with one end (source or use) not connected to any *box* in a *diagram*. Contrast: *internal arrow*.

2.1.15 boundary ICOM code: An *ICOM code* that maps an untunneled *boundary arrow* in a *child diagram* to an *arrow* attached to the *parent box* that is detailed by that *diagram*.

2.1.16 box: A rectangle containing a *box name*, a *box number*, and possibly a *box detail reference* andrepresenting a *function* in a *diagram*.

2.1.17 box detail reference: A square enclosure encompassing a *box number,* which indicates that the *box* is decomposed or detailed by a *child diagram*.

2.1.18 box ICOM code: An *ICOM code* that maps a tunneled *boundary arrow* to an *arrow* attached to some *ancestral box*.

2.1.19 box name: The verb or verb phrase placed inside a *box* that names the modeled *function*. A *box*takes as its box name the *function name* of the *function* represented by the *box*. See: *function name*.

2.1.20 box number: A single digit (***0, 1, 2, …, 9***) placed in the lower right corner of a *box* to uniquelyidentify that *box* in a *diagram*. The only *box* that may be numbered ***0*** is the *box* that represents the ***A0*** function in ***A-0*** and ***A-1*** *context diagrams*.

2.1.21 branch: A *junction* at which a *root arrow segment* (going from source to use) divides into two or more *arrow segments*. May denote *unbundling* of *arrow meaning*, i.e., the separation of *object types* from an *object type set*. Also refers to an *arrow segment* into which a *root arrow segment* has been divided.

2.1.22 bundle: (A) (As a verb) To combine separate *arrow meanings* into a composite *arrow meaning*, expressed by joining *arrow segments*, i.e., the inclusion of multiple *object types* into an *object type set*. (B) (As a noun) An *arrow segment* that collects multiple meanings into a single construct or abstraction, i.e., an *arrow segment* that represents an *object type set* that includes more than one *object type*.

2.1.23 call arrow: An *arrow* that enables the sharing of detail between *IDEF0 models* (linking them together) or within an *IDEF0 model*. The tail of a call arrow is attached to the bottom side of a *box*. One or more *page references* are attached to a call arrow.

2.1.24 call reference: A *page reference* attached to a *call arrow*.

2.1.25 called diagram: A *decomposition diagram* invoked by a *calling box* and identified by a *page reference* attached to a *call arrow*.

2.1.26 calling box: A *box* that is detailed by a *decomposition diagram* that is not the *box*'s *child diagram*. A *call arrow* is attached to the bottom of a calling box.

2.1.27 child box: A *box* in a *child diagram*.

2.1.28 child diagram: A *decomposition diagram* related to a specific *box* by exactly one child/parent relationship.

2.1.29 complete ICOM code: A *diagram feature reference* in which *dot notation* joins an *ICOM code* to a *diagram reference*.

2.1.30 context: The immediate environment in which a *function* (or set of *functions* in a *diagram*) operates.

2.1.31 context diagram: A *diagram* that presents the *context* of the *top-level function* of an *IDEF0 model*, whose *diagram number* is ***A-n,*** where ***0≤n≤9***. The one-box *A-0 context diagram* is a required context diagram; those with *diagram numbers* ***A-1, A-2***, …, ***A-9*** are *optional context diagrams*.

2.1.32 control: In an *IDEF0 model*, a condition or set of conditions required for a *function* to produce correct *output*.

2.1.33 control arrow: An *arrow* or *arrow segment* that expresses IDEF0 *control*, i.e., an *object type set* whose instances establish a condition or set of conditions required for a *function* to produce correct *output*. The arrowhead of a control arrow is attached to the top side of a *box*.

2.1.34 control loopback: *Loopback* of *output* from one *function* to be *control* for another *function* in the same *diagram*. Syn: *feedback*.

2.1.35 decomposition: The partitioning of a modeled *function* into its component *functions*.

2.1.36 decomposition diagram: A *diagram* that details its parent *box*.

2.1.37 descendent box: A *box* in a *descendent diagram*.

2.1.38 descendent diagram: A *decomposition diagram* related to a specific *box* by a *hierarchically consecutive* sequence of one or more child/parent relationships.

2.1.39 diagram: An instantiation of the formal diagram structure as defined by Annex B that consists only of semantically and syntactically valid IDEF0 graphical statements. Each diagram is a single unit of an *IDEF0 model* that presents the top-level *function* that is the subject of the model (the *A-0 context diagram*), presents the *context* of the subject *function* (other *context diagrams*), or presents the details of a *box* (*decomposition diagrams*).

2.1.40 diagram boundary: An edge of a *diagram* in a *diagram page*.

2.1.41 diagram feature: An element of a *diagram*. Diagram features include *boxes*, *arrow segments*, *arrow labels*, *ICOM codes*, *ICOM labels*, *model notes*, and *reader notes*.

2.1.42 diagram feature reference: An expression that unambiguously identifies a *diagram feature* within an *IDEF0 model*.

2.1.43 diagram number: That part of a *diagram reference* that corresponds to a *diagram*'s *parent function*'s *node number*. The diagram number refers to the *diagram* that details or decomposes the *function* designated by the same *node number*.

2.1.44 diagram page: A *model page* that contains a *context diagram* or a *decomposition diagram*.

2.1.45 diagram reference: An expression that unambiguously identifies a *diagram* and specifies the *diagram*'s position in a specific model hierarchy; a diagram reference is composed of a *model name abbreviation* and a *diagram number*.

2.1.46 diagram title: A verb or verb phrase that describes the overall *function* presented by a *diagram*; the diagram title of a *child diagram* is the *box name* of its *parent box*.

2.1.47 dot notation: A technique for naming that joins the name of a parent class to the name of a dependent class with the period character. For example, the *diagram feature reference* ***ABC/A31.3*** uses dot notation to join the *page reference* of the *parent diagram* ***ABC/A31*** to the *feature reference* for *box* ***3*** in that *diagram*.

2.1.48 feature reference: An expression that unambiguously identifies a *diagram feature* in a *diagram*.

2.1.49 feedback: See: *control loopback*.

2.1.50 FEO page: See: *For Exposition Only page*.

2.1.51 For Exposition Only page: A *model page* that contains pictorial and graphical information (in contrast to text) about a specific *diagram*. Unlike a *diagram*, the contents of a For Exposition Only page (*FEO page*) need not comply with IDEF0 rules.

2.1.52 fork: See: *branch*.

2.1.53 function: A transformation of *inputs* to *outputs*, by means of some *mechanisms*, and subject to certain *controls*, that is identified by a function name and modeled by a *box*. Syn: depending on the usage of a model, *activity*, task, process, operation.

2.1.54 function name: An active verb or verb phrase that describes what is to be accomplished by a *function*. A *box* takes as its *box name* the function name of the *function* represented by the *box*.

2.1.55 glossary: A set of definitions that includes *arrow labels* and *box names* used in an *IDEF0 model*.

2.1.56 glossary page: A *model page* that contains definitions for the *arrow labels* and *box names* in a specific *diagram*.

2.1.57 hierarchically consecutive: An unbroken unidirectional traversal of all *node*s between two specified *node*s in a tree. All *node*s between the origin and destination *node*s shall be visited during a traversal. All traversals from any *node* to its adjacent *node* shall be made in the same direction, either towards the root of the tree of towards the leaves of the tree. Typically, hierarchically consecutive is taken to imply from ancestral *node* (closer to the root) to descendent node (closer to the leaves).

2.1.58 ICOM code: An expression in one *diagram* that unambiguously identifies an *arrow segment* in another *diagram*. An ICOM code is used to associate a *boundary arrow* of a *child diagram* with an *arrow* attached to an *ancestral box*. Syn: *arrow reference*.

2.1.59 ICOM label: An *arrow label* attached without a *squiggle* directly to the arrowhead of an *output boundary arrow* or to the arrowtail of an *input*, *control*, or *mechanism boundary arrow*. An ICOM label associates a *boundary arrow* of a *child diagram* with an *arrow label* of an *arrow* attached to an *ancestral box*.

2.1.60 IDEF0 model: Abstractly, a hierarchical set of IDEF0 diagrams that depict, for a specific purpose and from a specific viewpoint, the functions of a system or subject area, along with supporting glossary, text, and For Exposition Only (FEO) information.

Concretely, a set of *model pages* that include at least an *A-0 context diagram* and an ***A0*** *decomposition diagram*, a *glossary* or specific *glossary pages*, one or more *text pages* to accompany each *diagram*, and *FEO pages* and *model pages* of other types as needed.

2.1.61 identifier: Within an *IDEF0 model*, a *model name*, a *box name*, or an *arrow label*.

2.1.62 input: In an *IDEF0 model*, that which is transformed by a *function* into *output*.

2.1.63 input arrow: An *arrow* or *arrow segment* that expresses IDEF0 *input*, i.e., an *object type set* whose instances are transformed by a *function* into *output*. The arrowhead of an input arrow is attached to the left side of a *box*.

2.1.64 input loopback: *Loopback* of *output* from one *function* to be *input* for another *function* in the same *diagram*.

2.1.65 internal arrow: An *arrow* connected at both ends (source and use) to a *box* in a *diagram*. Contrast: *boundary arrow*.

2.1.66 join: A *junction* at which an *arrow segment* (going from source to use) merges with one or more other *arrow segments* to form a *root arrow segment*. May denote *bundling* of *arrow meanings*, i.e., the inclusion of multiple *object types* within an *object type set*.

2.1.67 junction: A point at which either a *root arrow segment* divides into *branching arrow segments* or *arrow segments join* into a *root arrow segment*.

2.1.68 leaf diagram: A *diagram* that has no *descendent diagrams*, i.e., a *diagram* that does not contain any *function* that has been decomposed.

2.1.69 leaf node: A *function* that is not decomposed. A *box* that represents a leaf node does not have a *box detail reference*.

2.1.70 loopback: An *internal arrow* that is the *output* of a *box* whose *box number* is greater than the *box number* of the *box* that uses that *arrow* as *input*, *control*, or *mechanism*. These uses are referred to as *input loopback*, *control loopback*, and *mechanism loopback*, respectively.

2.1.71 mechanism: In an *IDEF0 model*, the means used by a *function* to transform *input* into *output*.

2.1.72 mechanism arrow: An *arrow* or *arrow segment* that expresses IDEF0 *mechanism*, i.e., an *object type set* whose instances are used by a *function* to transform *input* into *output*. The arrowhead of an mechanism arrow is attached to the bottom side of a *box*.

2.1.73 mechanism loopback: *Loopback* of *output* from one *function* to be *mechanism* for another *function* in the same *diagram*.

2.1.74 model hierarchy: The *diagrams* that correspond to the *node*s of the hierarchical graph structure of an *IDEF0 model*.

2.1.75 model name: A unique, descriptive name that distinguishes one *IDEF0 model* from other *IDEF0 model*s with which it may be associated. An *IDEF0 model*'s model name and *model name abbreviation* are placed in the *A-0 context diagram* along with the model's *purpose statement* and *viewpoint statement*.

2.1.76 model name abbreviation: A unique short form of a *model name* that is used to construct *diagram references*.

2.1.77 model note: A textual and/or graphical component of a *diagram* that records a fact not otherwise depicted by a *diagram*'s *boxes* and *arrows*.

2.1.78 model note number: An integer number, placed inside a small square, that unambiguously identifies a *model note* in a specific *diagram*.

2.1.79 model page: A logical component of an *IDEF0 model* that can be presented on a single sheet of paper. Model pages include *diagram*, *text*, *FEO*, and *glossary page*s.

2.1.80 node: A modeled *function* located within the hierarchical graph structure of an *IDEF0 model* by its designated *node number*; as a *function*, a node is represented in a *diagram* by a named *box*.

2.1.81 node index: A text listing, often indented, of the *node*s in an *IDEF0 model*, shown in outline order. Same meaning and *node* content as a *node tree*.

2.1.82 node letter: The letter that is the first character of a *node number*.

2.1.83 node number: An expression that unambiguously identifies a *function*'s position in a *model hierarchy*. A node number is constructed by concatenating a *node letter*, the *diagram number* of the *diagram* that contains the *box* that represents the *function*, and the *box number* of that *box*.

2.1.84 node tree: A graphical listing of the *node*s of an *IDEF0 model*, showing parent-child relationships as a graphical tree. Same meaning and *node* content as a *node index*.

2.1.85 object: A member of an *object set* and an instance of an *object type*. An *object* represents something in the observable world that may be distinguished from other instances of its *object type* and may be uniquely identified.

2.1.86 object set: A subset of instantiations from the set of all possible instantiations of all *object types* within an *object type set*. An object set is a subset of the union of the members of an *object type set*; the set of *object sets* includes the empty set and the set of the union of the members of the *object type set* itself. An object set is modeled by an *arrow segment*.

2.1.87 object type: The set of all possible instantiations of a singular concept, either physical or data, within an *IDEF0 model*. An IDEF0 object type is generally analogous to an IDEF1X entity or an IDEF1 entity class.

2.1.88 object type set: A named set of one or more *object types*. An object type set may include *object types* that are themselves grouped as object type sets. An object type set is designated by an *arrow label*.

2.1.89 output: In an *IDEF0 model*, that which is produced by a *function*.

2.1.90 output arrow: An *arrow* or *arrow segment* that expresses IDEF0 *output*, i.e., an *object type set* whose instances are created by a *function* by transforming the *function*'s *input*. The arrowtail of an output arrow is attached to the right side of a *box*.

2.1.91 page reference: An expression that unambiguously identifies a *model page*. The page reference incorporates a *diagram reference* to the associated *diagram*, the type of page, and any sequencing data needed to distinguish different pages of the same type that are associated with the same *diagram*.

2.1.92 page type letter: The uppercase letter in a *page reference* that denotes a specific type of *model page*.

2.1.93 parent box: An *ancestral box* related to its *child diagram* by exactly one parent/child relationship, that is, a *box* detailed by a *child diagram*. The existence of this *child diagram* is indicated by a *box detail reference*.

2.1.94 parent diagram: A *diagram* that contains a *parent box*.

2.1.95 parent function: A *function* modeled by a *parent box*.

2.1.96 purpose statement: A brief statement of the reason for an *IDEF0 model*'s existence that is presented in the *A-0 context diagram* of the model.

2.1.97 reader note: A comment made by a reader about an *diagram* and placed on the *diagram page*. A reader note is not part of the *diagram* itself, but rather is used for communication about a *diagram* during model development.

2.1.98 reference expression: An expression that uniquely identifies a *box*, a *node* or *function*, a *diagram*, or a *model page* within an *IDEF0 model*.

2.1.99 root arrow segment: The *arrow segment* of a *junction* from which other *arrow segment*s branch or to which other *arrow segment*s join. Syn: *root* or *root segment*.

2.1.100 root segment: See: *root arrow segment*.

2.1.101 squiggle: A short "s"-shaped line attached at one end to an *arrow label* and at the other end to an *arrow segment*. A squiggles binds an *object type set* (*arrow label*) to an *object set* (*arrow segment*).

2.1.102 text page: A *model page* that contains textual material related to a specific *diagram*.

2.1.103 top box: The *box* in the *A-0 context diagram* that models the *top-level function* of an *IDEF0 model*.

2.1.104 top-level function: The *function* modeled by the single *box* in the *A-0 context diagram* of an *IDEF0 model*.

2.1.105 tunneled arrow: An *arrow* left undrawn between its attachment to an *ancestral box* and its appearance as a *boundary arrow* on some *hierarchically consecutive descendent diagram*.

2.1.106 tunneling: The act of applying *tunnel notation* to an *arrow segment*.

2.1.107 tunnel notation: A pair of short shallow arcs, resembling a pair of left and right parentheses characters, that bracket the arrowhead or the arrowtail of an *arrow segment*.

2.1.108 unbundle: The separation of *arrow meanings*, expressed by branching *arrow segments*, i.e., the separation of *object types* from an *object type set*.

2.1.109 viewpoint statement: A brief statement of the perspective of an *IDEF0 model* that is presented in the *A-0 context diagram* of the model.

2.2 Acronyms and abbreviations

These short forms of words, names, and expressions are used in this document with the following meanings:

DRE	detail reference expression
ER	Entity relationship
FEO	For Exposition Only
ICOM	Input, control, output, and mechanism
IDEF	Integration Definition or ICAM Definition Language
IDEF0	IDEF for function modeling
IDEF1X	IDEF for data modeling
iff	if and only if
KIF	Knowledge interchange format
SDF	IDEF standard diagram form

3. References

No normative references are required for use with this standard. A bibliography of documents referenced in the body of the standard is provided in Annex A.

4. IDEF0 models

This clause and the clauses that follow identify the basic components of IDEF0 syntax (the drawn, visual elements of the language and how they may be used together) and IDEF0 semantics (what it means when the visual elements are used together in specific, allowable ways), specify the rules that govern the use of these modeling components, and describe the types of diagrams used in an IDEF0 model. Although the components of syntax and semantics are very highly interrelated, they are discussed separately without regard for the actual sequence of construction.

An *IDEF0 model* is a representation of a set of components of a system or subject area. Systems modeled by IDEF0 techniques are composed of interdependent parts that work together to perform an intended function. The system components represented in an IDEF0 model can be any combination of things, including people, information, software, functions, equipment, products, and raw materials. The IDEF0 model reflects how system functions interrelate and operate just as the blueprint of a product reflects how the different pieces of the product fit together.

An IDEF0 model describes what a system does, what controls it, what things it works on, what means it uses to perform its functions, and what it produces. An IDEF0 model is composed of a hierarchical series of diagrams with associated explanatory material that gradually introduce increasing levels of detail to describe functions and their interfaces within the context of a system.

An IDEF0 model contains four primary types of model pages: diagram, text, For Exposition Only (FEO), and glossary. Diagram pages contain diagrams that define functions and functional relationships in accordance with IDEF0 box and arrow syntax and semantics. Text, FEO, and glossary pages provide additional information in support of these diagrams. This structure provides the reader with a well-bounded topic with a manageable amount of detail to learn from a single diagram and with manageable amounts of supplementary information to learn from each of the diagram's supporting pages.

Concretely, an IDEF0 model shall contain a set of model pages that includes diagram pages that present at least the model's ***A-0*** context diagram and its ***A0*** decomposition diagram, one or more glossary pages for each diagram page, and one or more text pages for each diagram page. A glossary for the entire model may be used in lieu of individual glossary pages. FEO pages and model pages of other types defined for a particular modeling project may be included as needed.

5. IDEF0 syntax

This clause defines the visual aspects of boxes and their semantic interpretation as functions; the visual aspects of arrows and their semantic interpretation as objects, including data objects; and the visual aspects of attaching arrows to boxes and the semantic interpretation of these attachments as functional interrelationships.

The basic components of a language (i.e., the lexicon) and the rules that define how those basic components can be combined to form more complex syntactic structures (i.e., the grammar) together constitute the syntax of the language. The basic components of IDEF0 syntax are boxes and arrow segments.

5.1 Boxes

Boxes are the verbs of IDEF0 syntax: a *box* models a function in an IDEF0 model. A box shall be a rectangle with square corners. A box shall have a *box name* that is fully contained within its boundaries. The box name shall be an active verb or verb phrase that describes what must be accomplished by the function represented by the box. A box shall also contain a *box number* in its lower right corner. If a box has been detailed by a decomposition diagram, the box number shall be framed by an enclosed corner; this framing of a box number to indicate decomposition is known as a *box detail reference*[1]. A typical decomposed box is shown in Figure 1.

5.2 Arrows and arrow segments

Arrows are the nouns of IDEF0 syntax: an *arrow* models physical and data objects in an IDEF0 model. (The meaning of an arrow segment is further clarified in Clause 6.) An arrow is composed of one or more directed line segments, known as *arrow segments*, with a terminal arrowhead at one end of the arrow. As shown in Figures 2 through 5, an arrow segment shall be drawn as one or more straight horizontal and vertical line segments; an arrow may not be drawn diagonally. Horizontal and vertical line segments of an arrow segment shall be connected by a curved line segment with a 90° arc. Arrows may have branching and joining configurations.

An arrow is a linear configuration, in the sense that every two distinct arrow segments in an arrow are linked by some finite number of arrow segments. Thus, in Figure 5, the arrow ***I2:2I2*** is composed of the arrow segments labeled *Untrained Staff* and *Untrained Fitters* and the arrow segments *Trained Hefters* and *Trained Staff* constitute the distinct arrow ***3O1:O2***.

An *arrow label* consists of a single noun or noun phrase that designates the *object type set* of an arrow segment, that is, the physical and data objects represented by the arrow segment. Every arrow segment shall be labeled with an arrow label unless a single arrow label clearly applies to the arrow as a whole. Call arrows shall be labeled with one or more call references. A *squiggle* shall be used to link an arrow segment to its associated arrow label, unless a relationship between a label and a specific arrow segment is visually obvious. An arrow label that identifies a boundary arrow segment is attached to the arrowhead of an output boundary arrow or to the arrowtail of an input, control, or mechanism boundary arrow; such an arrow label is known as an *ICOM label*. Because the arrowhead or arrowtail of a boundary arrow is directly attached to the arrow label, the relationship between this label and the boundary arrow segment is always visually obvious and thus shall be drawn without a squiggle.

5.3 Names and labels

An IDEF0 model is identified by a model name, an IDEF0 box is identified by a box name, and an IDEF0 arrow segment is identified by an arrow label. These identifiers shall contain only alphanumeric characters, i.e., the letters (“A” through “Z” and “a” through “z”), the digits (“0” through “9”), the space character (“ ”), and the hyphen character (“-”, as in “well-defined”). Punctuation marks and other special characters shall not appear in identifiers. The only exceptions to this prohibition are characters required to represent a proper name such as the citation of a specific reference (e.g., *Roberts’ Rules of Order*, *DoD 8320.1-M-1*).

An identifier shall be written in title case, i.e., the first letter of each word shall be capitalized. The individual words of an identifier shall be separated by a single space; other characters (e.g., underscore (“_”), hyphen) shall not connect the words of an identifier to form a single token.

[1]A box detail reference may be associated with a *detail reference expression* (DRE) during the process of developing an IDEF0 model. A correct IDEF0 model does not require DREs.

Abbreviations shall not appear in an identifier. Prepositions (e.g., *for*, *to*, *from*), conjunctives (e.g., *and*, *or*), and articles (e.g., *the*, *an*) shall not appear in an identifier, unless such a word is an essential element of a term of art customarily used by a model's intended audience and the term of art is separately defined in the model's glossary.

An identifier

- Shall express a singular concept (hence the prohibition of conjunctives)
- Shall express the nature of the model component it identifies
- May not express a relationship between a model component and any other component(s) of a model (hence the prohibition of prepositions)
- May not refer to a specific instance of an object type or to a particular activation of a function (hence the prohibition of articles)

No identifiers for different arrows or boxes shall be identical.

(In the special case, limited to control and mechanism arrows that do not change roles, and where the object type set designated by an arrow label contain only one object type, and where the set of instantiations of the object type contains only one possible instance, and thus where the set of object sets consists only of the empty set and a set containing just this one instance, only one particular object can be instantiated from the object set, the object type, and the object type set. In this case, the name of the object type set is equivalent to the name of the instantiated object. Nonetheless, the arrow label still identifies the object type set represented by the arrow segment, not the single instance of the object set modeled by the arrow segment.)

a) *Names for boxes.* A box name shall identify the transformation represented by a box, i.e., the modeled function. A box name shall be an active verb or verb phrase such as

   - Order Parts
   - Monitor Performance
   - Develop Detail Design
   - Plan Resources
   - Design System
   - Fabricate Component

   A box name shall not contain any of these words: function, activity, or process.

b) *Labels for arrows.* An arrow label or ICOM label shall identify the object type set of an arrow segment. An arrow segment shall be labeled with a noun or noun phrase such as

   - Specifications
   - Performance Requirements
   - Software Engineer
   - Summary Report
   - Architectural Design
   - Board Assembly

   An arrow label shall not consist solely of any of these words: input, control, output, mechanism, call, object, or data.

Figure 6 illustrates the placement of arrow labels and box names.

5.4 Basic box/arrow syntax

More complex IDEF0 syntactic structures are formed by connecting boxes with arrows in various ways. Relative to a given box in a more complex diagram structure, there are five ways that an arrow may be attached to a box in an IDEF0 diagram. An arrow may be attached by its arrowhead to the top, to the bottom, or to the left side of a box. An arrow may be attached by its arrowtail to the bottom or to the right side of a box.

An arrow attached by its arrowhead to the left side of a box is called an *input arrow*. An arrow attached by its arrowhead to the top side of a box is called a *control arrow*. An arrow attached by its arrowhead to the bottom side of a box is called a *mechanism arrow*. An arrow attached by its arrowtail to the right side of a box is called an *output arrow*. An arrow attached by its arrowtail to the bottom side of a box is called a *call arrow*. These are the only valid ways to attach an IDEF0 arrow to an IDEF0 box. This basic box/arrow syntax is illustrated in Figures 7 through 9.

At least one control arrow and at least one output arrow shall be attached to a box. One or more input arrows and one or more mechanism arrows may be attached to a box. Only one call arrow may be attached to a box.

5.5 Call arrows

An important syntactic convention is the *call arrow*. Call arrows do not play a specific semantic role in an IDEF0 diagram; hence, they do not appear in Annex B. Rather, a call arrow allows a *calling box* to be detailed (i.e., an activity to be decomposed) by a diagram that is not an immediate child diagram of the calling box. A *called diagram* is a decomposition diagram that details a calling box. A calling box is said to *invoke* a called diagram. In other words, the details of an activity represented by a calling box are specified in a called diagram that is the immediate child of some other activity within the same model or possibly in another model entirely. A *call reference* is a page reference provided by a call arrow that identifies such a called diagram for a calling box.

a) *Model integration.* The call arrow convention allows the logical integration of models containing hierarchically related functions without requiring their integration into a single physical model. A call reference to a diagram in another model effectively links the model that contains the calling box to the model that contains the called diagram. For example, a call reference might link a box in a high-level context diagram of one model to the A0 diagram of another model to treat the subject matter of the linked activity.

b) *Shared detail.* Because more than one calling box may invoke the same called diagram via a call reference, the call arrow convention allows

   — Distinct functions with similar topologies to be modeled by a single generalized decomposition or

   — The same function to be identified in different places within the same model but to be only detailed in one place in the model's decomposition hierarchy

   This approach allows the sharing of decomposition detail by activities represented in different models and/or by activities within the same model. For example, the activities ***A323*** *Invite General Specifications Committee* and ***A5332*** *Invite Detail Specifications Committee* might both be detailed by diagram ***A5332***. A call arrow with the call reference ***A5332*** might be attached to box ***3*** in diagram ***A32***.

c) *Alternate activations.* Because a single calling box may invoke more than one called diagram via call references, the call arrow convention allows a set of detail diagrams to represent each different activation of a calling box. Each possible activation may be detailed by a called diagram that has

been prepared specifically to detail the activity under that particular activation, which may have been created in a different model or which details some activity elsewhere in the same model.

d) *Alternate representations.* Similarly, because a single calling box may invoke more than one called diagram via call references, the call arrow convention allows one function to be modeled by a set of alternative or variant detail diagrams. Each of these alternative decompositions may be represented by a diagram in a different model or by a detail diagram elsewhere in the same model.

A call arrow is a straight, single-segment arrow whose tail is attached to the bottom side of a calling box. Only one call arrow may be attached to a calling box. A call arrow is attached to the right of any mechanism arrows; mechanism arrows are always attached to the bottom side of a calling box to the left of any call arrow.

A call reference is a page reference that is attached to a call arrow to identify a called diagram that may be invoked by the calling box. Unlike arrow labels, a call reference may not be attached to the call arrow by a squiggle. A call reference shall be placed immediately above the back of the call arrow's arrowhead and immediately to the right of the call arrow segment, as shown in Figure 6.

More than one call reference may be attached to one call arrow. Should there be more than one call reference, the call references shall be left-justified within a vertical list. The last or bottom call reference in such a list shall be shown in exactly the same position relative to a call arrow's arrowhead as a single call reference. The order of call references in such a list has no significance. A calling box may not have a box detail reference because a calling box is not detailed by its own child diagram. A call reference may not be made to any kind of context diagram because a called diagram must be a decomposition diagram. For example, a calling box in a high-level context diagram for one model may incorporate another whole model only by invoking that model's ***A0*** diagram, which is the highest decomposition diagram in any IDEF0 model.

Different calling boxes may invoke the same called diagram and different called diagrams may be invoked by the same calling box. However, a calling box may invoke only one called diagram in any given activation of the calling box. Which activation conditions determine which called diagram is to be invoked by a calling box shall be specified in an accompanying model note. This note may refer the reader to a text page by a page reference for a complete explanation.

The boundary arrows of a called diagram need not correspond exactly with those of the calling box, either in number, name, or meaning. In these cases, a model note in the diagram containing the calling box shall specify these relationships so that a reader may appropriately interpret the shared physical and data objects required for an activation of the called diagram. This note may refer the reader to a text page by a page reference for a complete explanation.

6. IDEF0 semantics

Semantics involves, first, the interpretation of the basic syntactic components of a language and, second, how the interpretations of those basic components contribute to the meanings of more complex syntactic structures. Thus, a semantics for IDEF0 specifies how the boxes and arrows are to be interpreted and, in turn, how they contribute to the meanings of more complex structures built up from them.

6.1 Box/arrow semantics

a) *Box semantics.* A box models a *function*, i.e., an activity that, typically, takes certain inputs and, by means of some mechanism, and subject to certain controls, transforms its input into output. Thus, a *function name* shall be an active verb or verb phrase, such as "Inspect Parts," that describes what is accomplished by the function that the box represents. The name of a box is the name of the function represented by the box. Thus, for instance, a function named "Inspect Parts" would be

expected to transform uninspected parts into inspected parts, and the name of the box representing this function would also be “Inspect Parts.”

An IDEF0 transformation may consist of any combination of these four kinds of transform: form, state, time, and place.

1) A transformation of *form* substantively changes input objects into different output objects.

2) A transformation of *state* changes some characteristic(s) of input objects to produce the output objects. IDEF0 modeling treats something in one state before an IDEF0 transformation and the same thing in another state after the transformation as distinctly different objects.

3) A transformation of *time* changes the temporal location of input objects, i.e., moves input objects at one point in time to be output objects at another point in time, without necessarily changing their form or state. A transformation of time is the IDEF0 expression of storage.

4) A transformation of *place* changes the spatial location of input objects, i.e., moves input objects at one location to be output objects at another location, without necessarily changing their form or state. A transformation of place is the IDEF0 expression of movement or flow.

b) *Output without input.* An IDEF0 function must be provided control and must provide output. Input is not required for a syntactically correct statement of an IDEF0 activity (see 5.4). From the particular viewpoint and for the specific purpose of a given model, IDEF0 input for an activity may be omitted if all inputs meet one or more of these criteria:

1) *Control-bundled input*. An input is bundled with a control (see 6.6).

2) *Nonobject input*. An input is not a physical or data object. Not everything that is real can be modeled as an IDEF0 object. Real things such as emotions, knowledge, political power, and planetary orbits exist but cannot be represented by IDEF0 arrows.

3) *Nonconstraint input*. An input is always available to a function in all circumstances relevant to a given model. Such input—a precondition assumed always to be true—does not constrain the function.

4) *Creative act.* The function itself represents a creative act, a transformation that essentially creates something from nothing. As creative acts, functions such as *Write Program* and *Compose Sonata* are typically modeled as activities whose outputs are created without transformation of IDEF0 input.

c) *Arrow semantics.* At some level of abstraction, an IDEF0 arrow models physical and data objects that may exist in the tangible world and an arrow label identifies the kinds of objects that are represented by an arrow. However, because IDEF0 arrow segments may branch and join to commingle arrow meaning in arbitrary ways, distinctions among object type sets, object types, object sets, and objects themselves are necessary to allow an arrow segment to express the absence as well as the presence of objects and to allow consistent interpretation of branches and joins (see 6.2).

1) An *object type set* is a named set of one or more object types. An object type set may include object types that are themselves grouped as object type sets. An object type set is designated by an arrow label.

2) An *object type* is the set of all possible instantiations of a singular concept, either physical or data, within an IDEF0 model.

3) An *object set* is a subset of instantiations from the set of all possible instantiations of all *object types* within a named *object type set*. An object set is a subset of the union of the members of an object type set; the set of object sets includes the null or empty set and the set of the union of the members of the object type set. An object set is modeled by an arrow segment. Object sets are not explicitly named in IDEF0 syntax.

4) An *object* is one instantiation of an object type, that is, a member of an object set. A physical or data object represents something in the observable world that may be distinguished from other instances of its object type and may be uniquely named. However, an object is not explicitly named in IDEF0 syntax.

Each arrow segment shall be identified by the object type set of the object set modeled by the arrow segment. Generally, however, the terms *"an arrow represents an object type set"* or *"an arrow represents physical and data objects"* are used to express the relationship between an object type set, which is designated by an arrow label, and an object set, which is modeled by an arrow segment.

d) *Box/arrow attachments.* An arrow segment that is attached to a box represents an object type set in a specific role—input, control, mechanism, or output—for the function that the box represents. The side of a box to which an arrow is attached defines the role of an object type set with respect to the function represented by that box. (As noted in 5.5, call arrows play no semantic role but are merely pointers to diagram pages that may detail the calling box.)

1) *Input.* An arrow attached to the left side of a box represents input for the function. Such an input arrow represents what is transformed or consumed by the function to produce the function's output.

2) *Control.* An arrow attached to the top of a box represents control for the function. Such a control arrow represents conditions that must be met before the function can produce correct output.

3) *Output.* An arrow attached to the right side of a box represents output from the function. Such an output arrow represents what is produced by the function.

4) *Mechanism.* An arrow attached to the bottom of a box represents mechanism for the function. Such a mechanism arrow represents the means that carry out the function.

Input to an activity, i.e., an object set modeled by an arrow segment attached to the left side of a box, must be transformed by some activation of the function into some output of the activity, i.e., a different object set modeled by an arrow segment attached to the right side of that box. Output may result from transformation of inputs specified for an activity; output must account for all input and all input must be accounted for by output. Neither control nor mechanism may be transformed by a function. Thus, all input, control, output, and mechanism arrows represent object type sets, but only a control or mechanism arrow may represent an object type set that contains only one possible instantiation.

Arrows do *not* represent temporal sequencing as in a traditional process or data flow model; indeed, the concept of time cannot be formally expressed through IDEF0 syntax. Rather, arrows represent object type sets. These object type sets are necessary to specify a transformation represented by a box. The role that an object type set plays in this transformation is specified by attaching the appropriately labeled arrow to the appropriate side of the box. That one box is connected to another by an arrow simply specifies that the output of one function provides input, control, or mechanism needed for the activation of the other function.

Standard arrow attachments are shown in Figure 6.

6.2 Branching and joining arrows

Three or more arrow segments may connect at one point to form a *junction*. Such a junction is either a branch or a join. A *branch* splits one arrow segment, the root segment, into two or more arrow segments, the branching segments. A *join* combines two or more arrow segments, the joining segments, into a single arrow segment, their root segment. Branches and joins may be interpreted symmetrically with respect to their root segments. In both kinds of junction, the meaning of the root arrow segment—that is, the object

types represented by the root segment, without regard to how these object types may be grouped or otherwise bundled as an object type set—shall correspond to the object type sets represented by the arrow segments that connect to it.

Specifically, the meaning of the root segment of a join shall be equivalent to the *union* of the meanings of all arrow segments that join it. The meaning of a root segment of a branch shall be equivalent to the *union* of the meanings of all arrow segments that branch from it. The *union* of the meanings of all segments that join a root segment shall be equivalent to the *union* of the meanings of all segments that branch from the same root segment. Thus, additionally, the meaning of a root segment shall be equivalent to the *union* of the meanings of all arrow segments that connect to it. In other words, the *set* of arrow segments that connect in a junction to a root arrow segment must represent *all of and nothing but* the meaning of that root arrow segment.

This conservation of meaning of object types through arrow branching and joining applies to the *composite* meaning of a complete *set* of joining segments, the *bundled* meaning of their root segment, and the *composite* meaning of a complete *set* of segments branching from that root. This conservation of meaning of object types does not apply to the meaning of individual arrow segments that connect to a junction as joining or branching segments. While a *set* of connecting arrow segments, branching or joining, must represent *all of and nothing but* the meaning of the root segment of a junction, this conservation of meaning does not require that different object types of a root's meaning be apportioned among the segments that connect to the root segment in any particular way. This conservation of meaning *does* require each branching or joining arrow segment to represent a subset of the union of the members of the object type set represented by the root arrow segment of a junction.[2]

In general, the object type set of any individual arrow segment that connects to a root segment at a junction may contain any arbitrary combination of object types that are also contained within the meaning of the root segment, regardless of the meaning of any other segments that may also connect to the same root segment. However, two special cases frequently appear in IDEF0 models: fungible junctions and abstract junctions. These variations are illustrated in Figure 10.

a) In a *fungible junction*, exactly the same object type set is represented by every arrow segment that connects to the root segment. In this case, the separate meanings of the branching or joining segments will be exactly the same as the meaning of the root segment of the junction; the meaning of each connecting segment replicates the meaning of the root segment.

b) In an *abstract junction*, the meaning of the root segment is the *set addition* of the meanings of the other connecting segments. In this case, the *set addition* equals the *set union* of the meanings of the arrow segments that connect to a root segment in a junction. An abstract join *bundles* differentiated groups of physical and data objects into a generalized or more abstract grouping construct: the component parts, which are represented by joining arrow segments, are *bundled* into the root arrow segment. Such bundling is also termed *arrow aggregation*. Conversely, an abstract branch *unbundles* a generalized or more abstract grouping construct into differentiated or less abstract object type sets: the root segment is *unbundled* into its component parts, which are represented by branching arrow segments. Such unbundling is also termed *arrow decomposition*.

   When a junction is abstract, either join or branch, the meanings of the segments that connect to the root segment are disjoint, i.e., there is neither redundancy nor overlap of meaning among these arrow segments. In particular, when an abstract branch unbundles a root segment formed by an abstract join, the separate meanings of the branching segments will be the same as the separate meanings of the root's joining segments; in other words, the set of arrow labels identifying joining

[2]This conservation of meaning also determines that an arrow segment in an IDEF0 model may not represent anything that is not specified as input, control, output, or mechanism for some function within the scope of the model and that the set of all arrow segments in an IDEF0 model must represent everything that is required as input, control, output, and mechanism for all functions within the scope of the model.

segments must be the same as the set of arrow labels identifying branching segments. Because an arrow label specifies what an arrow segment represents by identifying an object type set, arrow labels on branching (joining) arrow segments may detail the content of a root arrow segment just as decomposition diagrams detail a parent box.

For example, given a root arrow segment that models *apples*, *oranges*, and *mangoes* as a bundle of *fruit*, any set of arrow segments that join into *fruit* or any set of arrow segments that branch from *fruit* must represent, in any combination, *apples* and *oranges* and *mangoes*; further, neither *bananas* nor *apricots* may be represented by any segment joining to or branching from this particular bundle of *fruit*.

An arrow segment by itself models an object set—a specific set of instantiations of the object types identified by the label attached to the arrow segment. The IDEF0 box/arrow syntax models such an object set in a specific role with respect to a given activity. In particular, any attachment of an arrow to a box as an input, control, or mechanism expresses an *activation constraint*, that is, a requirement for the existence of a nonempty object set in a particular role before a function can execute (see 6.7).

An arrow segment models an object set without enumerating or counting the members of the object set. Just as time is not formally represented by an IDEF0 model, the idea of quantity also cannot be formally represented through IDEF0 syntax. An arrow segment models objects in an object set without expressing any quantitative meaning.

Rather, the attachment of an input, control, or mechanism arrow to a box denotes that the function cannot activate unless sufficient instances of the arrow's object types (that is, an object set) are available to the function. What "sufficient" might mean is not determined in an IDEF0 model by characterizing an arrow. Whatever "sufficient" might mean in a counting sense, "sufficient" is expressed in an IDEF0 model by the structure of IDEF0 controls, which prevent activation of a function until sufficient resources of the appropriate kinds are available and which stimulate activation of other functions until these sufficient resources are produced.

Thus, in an IDEF0 model, objects are not quantified or counted; the significance of this absence of quantity is that conservation of arrow meaning through junctions, i.e., through joins and roots and branches, does not require or imply that an arrow segment represent any specific number of objects. While IDEF0 box/arrow syntax can be used to specify, say, that *apples*, *oranges*, and *mangoes* are required to activate a function, this syntax cannot be used to specify *how many* apples, *how many* oranges, or *how many* mangoes are required. Thus, conservation of arrow meaning does not imply that the *number* of objects represented by a set of segments branching from a root segment conserves the *number* of objects represented by a corresponding set of segments joining that root.

6.3 Arrow meaning conventions

Because the meanings of arrow segments connected in a junction may be ambiguous, IDEF0 provides several conventions to clarify these meanings. These conventions also minimize the number of arrow labels that need be physically provided in a diagram. The conventions and their exceptions are illustrated in Figures 12 through 21. The conventions are

a) The meaning of any boundary arrow, expressed by its ICOM label, may be propagated to all unidentified connecting arrow segments, both joining and branching, unless this propagation is halted by

   1) A *labeled segment* (i.e., an arrow segment that is explicitly identified by an arrow label or an ICOM label),

   2) An *ambiguous segment* (i.e., one arrow segment that can be implicitly identified by different arrow labels via label propagation along different arrows), or

3) *Ambiguous attachments* (i.e., different arrow segments, all attached to the same box, that can be implicitly identified by the same arrow label via label propagation along different arrows).

In particular, an ICOM label for an input, control, or mechanism boundary arrow may identify all unidentified branching arrow segments, including subsequent arrow segments of that boundary arrow and arrow segments that branch from it. In addition, an ICOM label for an output boundary arrow may identify all unidentified joining arrow segments, including antecedent arrow segments in that boundary arrow and arrow segments that join it.

b) After ICOM labels of boundary arrow segments have been propagated, the arrow labels attached to other root arrow segments may be similarly propagated to branching arrow segments. The meaning of such root arrow segments, expressed either explicitly by an arrow label or implicitly by propagation of an ICOM label, may be propagated through all connected arrow segments, unless, as with boundary arrow ICOM labels, such propagation is halted by a labeled segment, an ambiguous segment, or ambiguous attachments.

Only ICOM labels and then arrow labels attached to the root segment of a branch may be propagated. In particular, an explicit arrow label attached to an arrow segment formed by joining arrow segments may not be propagated to those joining arrow segments. In contrast, if such a root segment may be implicitly identified by propagation of an ICOM label, the implicit identity of the bundled root segment may propagate to other, unidentified joining arrow segments.

6.4 Ambiguous arrow segments

The meaning of an arrow segment may propagate to another arrow segment only if it is the only meaning that may be propagated to that arrow segment. An ambiguous arrow segment arises if the meaning of the same arrow segment can be differently implied by the propagation of different labels along different arrows. An ambiguous arrow segment shall be explicitly labeled and may not be implied. A label may neither be propagated to nor through an ambiguous segment. In particular,

a) The meaning of an arrow segment may not propagate to an unidentified arrow segment that connects two arrow segments by branching from one and joining to the other, whether labeled explicitly or implicitly by propagation.

b) The meaning of an arrow segment may not propagate to an unidentified arrow segment that results from the join of arrow segments that have different meanings. Similarly, the meaning of an arrow segment may not propagate to an unidentified arrow segment that is the root of branching arrow segments that have different meanings, whether labeled explicitly or implicitly by propagation.

6.5 Ambiguous arrow attachments

The meaning of an arrow segment may propagate to only one arrow segment attached to a given box. Ambiguous arrow attachments arise if the meaning of an arrow segment can be propagated along different arrows to multiple arrow segments attached to the same box. An arrow segment with an ambiguous arrow attachment shall be explicitly labeled and may not be implied. A meaning may not be propagated to an arrow segment with an ambiguous attachment. In particular,

a) The meaning of an arrow segment may not propagate to any output arrow segment attached to a given box if that meaning can also propagate to any input, control, or mechanism arrow segment attached to the same box. An arrow segment meaning may propagate to an output arrow segment if and only if that arrow segment label cannot also propagate to any input, control, or mechanism arrow segment attached to the same box.

b) The meaning of an arrow segment may not propagate to any output arrow segment attached to a given box if that meaning can also propagate to any other output arrow segment attached to the same box. An arrow segment meaning may propagate to an output arrow segment attached to a

given box if and only if that arrow segment label cannot also propagate to any other output arrow segment.

c) The meaning of an arrow segment may not propagate to any input, control, or mechanism arrow segment attached to a given box if that meaning can also propagate to any other input, control, or mechanism arrow segment attached to the same box. An arrow segment meaning may propagate to an input, control, or mechanism arrow segment attached to a given box if and only if that arrow segment label cannot also propagate to any other input, control, or mechanism arrow segment.

6.6 Arrow role conventions

A significant IDEF0 graphical convention allows a single control arrow to represent both control *and* input for the same box if these arrow segments both branch from the same root. In this exceptional convention, roles rather than objects are bundled in the control arrow. This convention reduces the complexity of a diagram and emphasizes the control role of the bundled arrow; these features are displayed in Figure 11. An arrow attached to an ancestral box as a control can appear in a descendent diagram as a control, as an input, or as both a control and an input, depending on the relationship of the bundled arrow to the activities modeled in the descendent diagram. When this convention is used, the detail of separate input and control roles shall be shown in a descendent diagram. The diagram exhibiting this detail shall show the unbundling of these arrow roles into separate inputs and controls and their attachments to different boxes for different purposes.

An object may have more than one role with respect to an activity; the role is not inherent in the nature of the object. This convention allows two roles to be represented by one arrow for convenience and to reduce diagram clutter. This convention does *not* change control semantics into input semantics: an object in the role of control may *not* be transformed into output, while an object in the role of input *must* be transformed into output.

6.7 Activations

Models are general descriptions: they describe the general structure of a given process, the pattern exhibited by any given instance, or *activation*, of that process. It is very important to note that there need not be a contribution from every input, control, and mechanism arrow attached to a box in every activation of the function represented by the box, nor need there be output of the sort indicated by every output arrow. Under different circumstances, a box may perform selected parts of its function using different combinations of input, control, and mechanism to produce different output. Hence, in general, a diagram represents many different possible activations of any modeled process.

In particular, any attachment of an arrow to a box as input, control, or mechanism expresses an *activation constraint*, that is, a requirement for a nonempty object set in a particular role as a precondition to some execution of the activity; the function cannot act to produce some output in the absence of objects specified by the nonempty object set. Which activations of a function are so constrained is determined by the decomposition diagrams that detail the box to which the activation constraint applies.

6.8 Concurrent activation

Several functions in a model may be performed concurrently, if the needed constraints have been satisfied. As illustrated in Figure 22, output of one box may provide some or all the physical and data objects needed for activations of one or more other boxes. When output of one box provides some or all the input, control, or mechanism needed by another box, a given activation of the latter box may depend on sequential performance.

7. IDEF0 diagrams

Several types of diagrams exist within an IDEF0 model. This clause identifies and discusses the types of IDEF0 diagrams and their relationships.

The graphic IDEF0 *diagram* is the primary component of an IDEF0 model. An IDEF0 diagram contains named boxes and labeled arrows; a diagram structures these graphic elements in accordance with the syntactic and semantic rules of IDEF0. A box represents a function (see 6.1). A function may be analyzed or *decomposed* into a more detailed diagram, and the boxes in such a decomposition diagram may in turn be further decomposed into diagrams providing additional detail. Such decomposition continues until a subject is rendered at the level of detailed exposition necessary to support the stated purpose of a particular model.

The starting point for this analysis is the ***A-0*** *context diagram*, which provides the most general or abstract description of the subject represented by a model. The ***A-0*** context diagram introduces a series of one or more *decomposition diagrams*, starting with the ***A0*** diagram, which provide successively deeper analysis of the modeled subject.

The ***A-0*** context diagram also defines the immediate context of the ***A0*** function of an IDEF0 model; *optional context diagrams* may provide further environmental context to expand and augment this immediate context. These optional context diagrams include the ***A-1*** *context diagram*.

All decomposition diagrams in an IDEF0 model *descend* from the model's ***A-0*** context diagram. Conversely, all context diagrams in an IDEF0 model *arise* from the model's ***A0*** decomposition diagram.

7.1 Diagram identification

A diagram is identified within an IDEF0 model by both a unique diagram number (see 10.3) and a unique diagram title. The diagram number and the diagram title that identify a diagram shall be recorded on the diagram page that contains the identified diagram. With the exception of the ***A-0*** context diagram and of the highest optional context diagram, the diagram title shall be the same as the box name of the diagram's parent box and the diagram number shall be the same as the node number of the diagram's parent function.

Neither the ***A-0*** context diagram nor the highest optional context diagram has a parent box from which to inherit its diagram title nor a parent function from which to inherit its diagram number. A diagram title for these diagrams shall combine the phrase "Context of" with a substantive equivalent of the box name of the most important box in the diagram. For example, given an ***A-0*** context diagram whose single box contains the box name "Build Ship," the diagram title of this context diagram would be "Context of Ship Building." In contrast, the diagram numbers for these context diagrams are a part of the IDEF0 language itself (see 10.3).

7.2 A-0 context diagram

An IDEF0 model shall include an ***A-0*** context diagram (see Clause 4) to be parent to the model's ***A0*** decomposition diagram. This diagram identifies the model, determines the subject of the model, and defines the scope of analysis to be included in the model. This diagram also defines the model's interfaces with other activities that are outside the scope of the model. This context diagram is always called the "A-0 context diagram," where "A-0" is pronounced "A minus zero," regardless of the actual node letter used in a model (see 10.2).

The *only* function shown in an ***A-0*** context diagram is the ***A0*** function of a model. In the ***A-0*** context diagram of a model, the ***A0*** function of the model is represented by box ***0***. This ***A0*** function represents the whole of the subject of the model; it is the unique parent of the entire modeled subject and thus the

ancestor of all activities modeled. By representing the ***A0*** function, the ***A-0*** context diagram sets a model's scope, that is, the boundaries of what may be included in that model.

The necessary and sufficient context required to understand and to analyze the ***A0*** function according to the model's viewpoint and purpose is given by boundary arrows that identify inputs, controls, and mechanisms required by the ***A0*** function and outputs that it produces. The inputs, controls, and mechanisms for the ***A0*** function are made available by activities that are outside the scope of a model. Similarly, the outputs of the ***A0*** function are used by activities that also are outside the scope of a model. These external activities shall not be analyzed or represented in the decomposition diagrams of a model. Because the context for the ***A0*** function as given by the ***A-0*** diagram does not identify these external functions, this context is often called the *immediate* context of the ***A0*** function.

All decomposition diagrams are strictly bound in scope to the limits of the ***A0*** function as specified by the ***A-0*** context diagram. In particular, the boundaries of a model's ***A0*** decomposition diagram are precisely coextensive with the boundaries of the model's ***A-0*** context diagram. Thus, an ***A-0*** context diagram is the strict parent of its ***A0*** decomposition diagram and the ancestor of all further decomposition diagrams.

The ***A-0*** context diagram of an IDEF0 model shall contain only one box, box ***0***, which represents the ***A0*** function of the model. The ***A-0*** context diagram shall present the model name, the model name abbreviation, and brief text statements that specify the viewpoint and the purpose of the model; these statements will guide and constrain the model's development. Boundary arrows connected to box ***0*** shall identify the necessary and sufficient inputs, controls, mechanisms, and outputs required to establish the immediate context of the ***A0*** function. These boundary arrows may not have ICOM codes or tunnel notations at their unconnected ends.

All decomposition diagrams in an IDEF0 model *descend* from the model's ***A-0*** context diagram. Conversely, all context diagrams in an IDEF0 model *arise* from the model's ***A0*** decomposition diagram. Due to its unique responsibility to define the purpose and scope of an IDEF0 model, an ***A-0*** context diagram does not itself have a parent diagram; instead, the parental context of the ***A-0*** diagram is defined by the IDEF0 language to be the token **TOP** (see also 7.7).

An example ***A-0*** context diagram is shown in Figure 23.

7.3 Model name

Each model shall be given a unique, descriptive *model name* that distinguishes it from other models. A unique *model name abbreviation* shall be derived from this model name; the model name abbreviation is used to construct diagram reference expressions.

A model's full name and its abbreviation shall be placed together in the ***A-0*** context diagram of the IDEF0 model. A model name abbreviation shall consist of two or three uppercase alphanumeric characters. For example, a model named "Manufacturing Operations" may be abbreviated **MFG** (see 10.6 for a discussion of diagram references). A model name abbreviation is typically given in parentheses following the model name, in this way: "Maritime Pipe Laying (MPL)." A model name is a required textual element in an ***A-0*** context diagram, and, therefore, shall not be marked as a model note. Each model shall have only one model name and one model name abbreviation.

7.4 Model viewpoint

A *viewpoint statement* guides and constrains a model's development. This statement determines what can be seen within a model's context and shapes what a model will notice and pay attention to. Different viewpoints emphasize different aspects of the same subject and necessitate different models. Functions, objects, and data that are important in one viewpoint may not even appear in a model developed from another viewpoint of the same subject.

The viewpoint statement for a model shall be placed in the ***A-0*** context diagram of the model and is a brief sentence that identifies a person or a personified role. The perspective of this person or role determines the model's viewpoint. A viewpoint statement is a required textual element in an ***A-0*** context diagram, and, therefore, shall not be marked as a model note. Each model shall have only one viewpoint statement.

7.5 Model purpose

A *purpose statement* guides and constrains a model's development and expresses the reason a model is created. The purpose statement determines the question that is to be answered by a model. When the purpose statement is fulfilled by a satisfactory answer to the question posed, the modeling effort concludes.

The purpose statement for a model shall be placed in the ***A-0*** context diagram of the model and is a brief sentence that identifies the question addressed by a model. A purpose statement is a required textual element in an ***A-0*** context diagram, and, therefore, shall not be marked as a model note. Each model shall have only one purpose statement.

7.6 Optional context diagrams

An IDEF0 model may include context diagrams (see 7.2) that provide a fuller exposition of the environment of a model. These optional diagrams examine the environment of the ***A0*** function and model a *broader* scope than the ***A-0*** context diagram. Because optional context diagrams incorporate functions that are outside the scope of a model's decomposition diagrams, these diagrams are often called the *environmental* context of the ***A0*** function.

By providing a more extensive description of a modeled system's environment, optional context diagrams provide more constraining specifications on the boundary conditions of a modeled system. Contextual modeling provides details about the sources and uses of the physical and data objects modeled as boundary arrows in the ***A-0*** context diagram. Such detail may not precisely match any particular "real world" environment but this detail is not intended to be definitive. Optional context diagrams serve their purpose sufficiently by describing typical environmental contexts.

Optional context diagrams add functions to the environment of the ***A0*** function of a model. These functions are outside the scope of the ***A0*** diagram as established by the ***A-0*** context diagram. The boundary arrows of the ***A-0*** context diagram specify the inputs, controls, and mechanisms required by the ***A0*** function and the outputs it produces. These neighboring functions provide the ***A0*** function with these inputs, controls, and mechanisms. These functions can also use outputs of the ***A0*** function for their own activations. Optional context diagrams illuminate this set of mutually constraining dependencies and so define an environment that is necessary for the ***A0*** function to operate.

— *The A-1 context diagram.* The first of the optional context diagrams is the ***A-1*** context diagram. The ***A-1*** context diagram (pronounced "A minus one") is constructed to be a second parent to a model's ***A0*** decomposition diagram (see also 7.2). This parent/child relationship between the ***A-1*** and the ***A0*** diagrams links the optional context diagrams to the decomposition diagrams that descend from the ***A-0*** context diagram. Because the ***A0*** function appears in ***both*** the ***A-0*** and ***A-1*** context diagrams, ***both*** context diagrams are parents of the model's ***A0*** diagram. If present, the ***A-1*** context diagram replaces the ***A-0*** context diagram as the parental context of the ***A0*** diagram.

— *Lineal ancestry.* Optional context diagrams include *high-level* context diagrams and their contextual descendants. High-level context diagrams are lineal ancestors of the ***A0*** diagram because they decompose the lineal ancestors of the ***A0*** function. Their contextual descendants decompose other functions that are not lineal ancestors of the ***A0*** function. The *high-level* context diagrams ***A-1***, ***A-2***, ***A-3***, ***A-4***, ***A-5***, ***A-6***, ***A-7***, ***A-8***, and ***A-9*** are defined as the *lineal ancestors* of the

A0 diagram. The ***A-1***, ***A-2***, ***A-3***, ***A-4***, ***A-5***, ***A-6***, ***A-7***, and ***A-8*** functions are defined as the *lineal ancestors* of the ***A0*** function.

The ***A-1*** context diagram is one parent of the ***A0*** diagram (the other parent is the ***A-0*** context diagram). The grandparent of the ***A0*** diagram is the ***A-2*** context diagram, the great-grandparent of the ***A0*** diagram is the ***A-3*** context diagram, and so on, through the ***A-9*** context diagram, which is the highest ancestry allowed. (An ***A-1*** context diagram is *not* parent to the ***A-0*** context diagram.)

A model may contain up to nine lineal ancestors to the ***A0*** diagram as optional context diagrams. The last ancestral diagram in a sequence of lineal ancestors will have no parent diagram and is called the *highest* optional context diagram. All parent/child diagram relationships between the highest optional context diagram and the ***A0*** diagram must be included in a model. Thus, if the highest optional context diagram is the ***A-5*** context diagram, then the ***A-1***, ***A-2***, ***A-3***, ***A-4***, and ***A-5*** context diagrams must all be included in the model.

Figure 25 illustrates, in node-tree form, the diagram structure of a possible environmental context for the ***A0*** function of an IDEF0 model. Both the ***A-0*** and the ***A-1*** context diagrams are parents to the ***A0*** diagram. The ***A-2*** and ***A-1*** context diagrams are lineal ancestors of the ***A0*** decomposition diagram. The ***A-2*** diagram is the grandparent of the ***A0*** diagram and the highest context diagram in this model. The ***A-24*** and ***A-212*** context diagrams are contextual descendants of the lineal ancestor ***A-2***. The ***A-13*** and ***A-141*** context diagrams are contextual descendants of the lineal ancestor ***A-1***. Note that all context diagrams are characterized by negative diagram numbers. Note also that the ***A-2*** diagram is parent to diagrams ***A-21***, ***A-22***, ***A-1***, and ***A-24***. By the rules that associate box numbers, node numbers, and diagram numbers, there must be boxes with box numbers ***1***, ***2***, ***-1***, and ***4*** in diagram ***A-2***. Diagrams ***A-21***, ***A-22***, and ***A-24*** are contextual descendants of the ***A-2*** diagram while diagram ***A-1*** is a lineal ancestor of the ***A0*** diagram.

A model may contain any number of optional context diagrams; functions shown in optional context diagrams may be decomposed in the ordinary way (see 7.7). A function that provides or uses physical and data objects modeled as boundary arrows in the ***A-0*** diagram may be represented by a box in any optional context diagram. The environmental context is not required to identify functions to provide every input, control, or mechanism or to use every output defined in the ***A-0*** diagram. A boundary arrow in the ***A-0*** diagram may also be represented as a boundary arrow in the highest optional context diagram, and such representation indicates that the source or use of this physical or data object is not examined within the presented environmental context.

Optional context diagrams shall follow the same syntactic and semantic rules as decomposition diagrams, with these exceptions:

a) Diagrams, functions, and boxes that represent the lineal ancestry of the ***A0*** function shall be numbered using predefined negative numbers (see Clause 10).

b) In each optional context diagram that is a lineal ancestor of the ***A0*** diagram, one box shall be assigned a negative box number.

c) A function that is a lineal ancestor of the ***A0*** function, i.e., a function identified in an optional context diagram by a predefined negative node number, shall be decomposed.

d) A box with a negative box number represents a function that is a lineal ancestor of the ***A0*** function. Such a box shall have a box detail reference.

e) Arrows shall not be tunneled in optional context diagrams. Each arrow attached to a parent box shall correspond to a boundary arrow on the child diagram. Arrow tunneling is not permitted in any optional context diagrams, except to show tunnel notation that may have been used in the ***A-0*** diagram.

Additional requirements for the ***A-1*** context diagram are

— The ***A-1*** context diagram shall contain one box with the box number ***0*** to represent the ***A0*** function of a model (see 10.1).

— The immediate context of the ***A0*** function as specified in the ***A-0*** diagram shall be repeated in the ***A-1*** diagram, that is, all physical and data objects modeled by boundary arrows attached to box ***0*** in the ***A-0*** context diagram shall be also be represented by arrows attached to box ***0*** in the ***A-1*** context diagram. Only those arrows shown in the ***A-0*** context diagram may be attached to box ***0*** in the ***A-1*** context diagram; no additional arrows may be attached to box ***0*** in the ***A-1*** diagram.

An example ***A-1*** context diagram is shown in Figure 24.

Additional requirements for the highest optional context diagram are

— The parental context of the highest optional context diagram is defined by the IDEF0 language as the token **NONE**.

— As with the ***A-0*** diagram, boundary arrows in the highest optional context diagram may not attach to ICOM codes.

7.7 Decomposition diagrams

Decomposition is the partitioning of a modeled function into its component functions. A *decomposition diagram* is a diagram that shows the detail of boxes and arrows for such a decomposition. The single function represented in the ***A-0*** context diagram is decomposed into its major subfunctions by creating its decomposition diagram. In turn, each of these subfunctions may be decomposed, each creating another lower-level decomposition diagram. A decomposition diagram is also called a *child diagram*, and the diagram that contains the box that is detailed by the decomposition is also called a *parent diagram*. Some of the functions, none of the functions, or all of the functions in a given diagram may be decomposed. Each decomposition diagram contains the boxes and arrows that provide additional detail about the decomposed box.

The diagram that results from the decomposition of a function covers the same scope as the box it details. Thus, a decomposition or child diagram may be thought of as the interior of the parent box it details. This decomposition structure is illustrated in Figure 26.

7.8 Parent/child diagram relations

A *parent diagram* is a diagram that contains one or more *parent boxes*, i.e., a box that is detailed by a decomposition diagram. Every ordinary (noncontext) diagram is also a child diagram, since by definition it details a parent box. Thus a diagram may be both a parent diagram (containing parent boxes) and a child diagram (detailing its own parent box). Likewise, a box may be both a parent box (a box detailed by a child diagram) and a *child box* (a box that appears in a child diagram). The primary hierarchical relationship is between a parent box and the child diagram that details it.

Thus ***A0*** is always the node number of the top-level function in a model and at the same time the diagram number of its child diagram; together, these represent the whole model. The model's parent ***A0*** function is always detailed by functions with node numbers ***A1***, ***A2***, ***A3***, to at most ***A9***.

The parental context provides or names the context for a diagram without its own parent diagram. If there is no ***A-1*** context diagram, the parental context of the ***A0*** diagram is the ***A-0*** context diagram. If there is an ***A-1*** context diagram, the ***A-1*** diagram is taken to be the parent of the ***A0*** diagram.

A box that is not further decomposed is called a *leaf node* of the decomposition. A leaf node is represented as a box that does not have the enclosed corner known as a box detail reference.

8. IDEF0 model pages

Each diagram in an IDEF0 model is associated with several model pages. These pages are identified and discussed in this clause.

A model page is associated with and provides information supporting a specific diagram. A model page is cross-referenced to other model pages by page references, which tie each model page to its associated diagrams. A page reference for a model page is formed by appending one or more page type letters to the diagram reference of an associated diagram (see 10.7).

A model page that provides general or global information for an entire model rather than for one specific diagram shall be associated with the ***A-0*** context diagram.

A model page may be presented on an IDEF0 standard diagram form (SDF).

8.1 Diagram pages

A diagram page is a model page that contains an IDEF0 context or decomposition diagram. Each diagram in an IDEF0 model shall be presented on a separate diagram page. A diagram page may be designated in a page reference by the page type letter "**D**."

8.2 Text pages

A text page is a model page that contains information primarily presented in words rather than pictures or graphics. A text page may supplement a model page of any type. A text page shall be designated in a page reference by the page type letter "**T**."

At least one text page is required to describe each diagram in a model. This text page shall provide a concise overview of the diagram. Text shall be used to highlight features, structures, and interbox connections and to clarify the intent of items and patterns considered to be of significance. Text shall not be used merely to describe, redundantly, the meaning of boxes and arrows.

8.3 Glossary pages

A glossary page is a model page that contains terms and their definitions. A glossary page may supplement a model page of any type and shall be designated in a page reference by the page type letter "**G**."

a) *Model glossary.* Instead of individual glossary pages, a single alphabetically sorted glossary may be provided for all terms used an IDEF0 model.

b) *Glossary entries.* A glossary entry shall include the text of the term to be defined and the text of its definition. The text of a glossary term shall be identical to the text of the term as it appears in the diagrams of the model.

   Each arrow label in a model shall be defined as a glossary entry. The definition for an arrow label shall include a description of the content of the arrow segment that the arrow label identifies, i.e., the arrow meaning. If an arrow segment is a bundled collection of other arrows, the definition shall identify the arrows in the bundle.

   Each box name in a completed model that corresponds to a leaf node shall be defined as a glossary entry. The definition for a leaf node's box name shall describe the function represented by the box, as though the box had been decomposed.

A box name that corresponds to a decomposition diagram (i.e., a nonleaf node) may be included as a glossary entry. The definition for a box detailed by a child diagram shall be of the form "See (diagram number of the function's decomposition)." For example, the definition for some function ***A32** Prepare Glossary Definitions* that is the parent of a decomposition diagram ***A32***, would be: "Prepare Glossary Definitions. See diagram A32."

Terms that may be useful for understanding a diagram (e.g., key words, phrases, acronyms) or terms used to construct arrow labels and box names (e.g., constituent terms) may be included as glossary entries.

8.4 FEO pages

A FEO (pronounced "fee-oh") page is a model page that contains information primarily presented in pictures or graphics rather than words. A FEO page may supplement a model page of any type and shall be designated in a page reference by the page type letter "**F**."

As required, FEO pages may be used to supplement a diagram. A FEO page shall be used where additional graphically oriented information is required to adequately understand a specific area of a model. Such supplementary information should be limited to what is needed to achieve the stated purpose of a diagram for a knowledgeable audience. The contents of a FEO page need not comply with IDEF0 syntax rules.

8.5 Other pages

As required, other types of pages may be defined with a meaning and usage specific to a particular model, modeling process, or modeling project, e.g., kit pages within a model construction kit cycle.[3] A page type letter shall be assigned for each such type of page; this page type letter shall then be used to create page references for model pages of that type.

9. IDEF0 diagram features

This clause discusses specific features found in an IDEF0 diagram, including box and arrow layouts that have syntactic significance, ICOM coding, arrow tunneling, and model notes.

9.1 Boxes

Except for the ***A-0*** context diagram, a diagram shall contain a minimum of two and a maximum of nine boxes. Boxes are normally organized diagonally from the upper left corner to the lower right corner, i.e., in a configuration resembling a staircase descending to the right. This diagonal organization of boxes allows input, control, output, and mechanism (ICOM) boundary arrows to be drawn directly to each box without routing around other boxes. An arrow and a box shall not occupy the same space nor overlay each other in a diagram; an arrow that does not attach to a box shall go around the box.

9.2 Interbox connections

Any output arrow may provide some or all of the input, control, or mechanism physical or data objects to any other box. An output arrow may provide physical and data objects to several boxes by branching, as shown in Figure 27.

[3]The IDEF kit cycle is outside the scope of this standard.

If a box in a diagram is detailed by a child diagram, each arrow connected to the parent box shall appear in the child diagram, unless the arrow is tunneled at a side of the parent box (see 9.4). The arrow labels of arrows attached to a parent box shall be identical to the arrow labels of the boundary arrows in its child diagram.

Each box in a diagram shall be connected by some sequence of arrows and boxes to at least one control boundary arrow and to at least one output boundary arrow in that diagram. An unconnected box may not appear in a diagram.

In a diagram, physical and data objects are represented by internal arrows and by boundary arrows. An *internal arrow* is an arrow that is attached at both head and tail to boxes in the same diagram. An internal arrow represents the output of one function that is used as the input, control, or mechanism for another function in the same diagram. In contrast, a *boundary arrow* is an arrow that is attached at only one end, either head or tail, to one box in a diagram; the unconnected end represents a connection to an arrow segment attached to a box in an ancestral diagram. Internal arrows and boundary arrows are shown in Figure 28. Boundary arrows are discussed in detail in 9.3.

Loopback refers to an internal arrow that is the output of a box whose box number is greater than the box number of the box that uses that arrow as input, control, or mechanism. As illustrated in Figure 29, control loopback shall be drawn as “up and over.” As illustrated in Figure 30, both input and mechanism loopback shall be drawn as “down and under.”

9.3 Boundary arrow segments

Boundary arrows on a decomposition diagram represent the inputs, controls, outputs, and mechanisms of the diagram’s parent function. The source or use of these boundary arrow segments can be found only by examining the parent diagram. Each boundary arrow in a child diagram shall have the same name as exactly one arrow segment attached to an ancestral box of that diagram, as shown in Figure 31.

There shall be a one-to-one correspondence between the arrow segments attached to a parent box and the boundary arrow segments of its decomposition diagram. Input, control, or mechanism boundary arrows that represent the same physical and data objects shall be connected through a branch; the boundary arrow segment shall branch into as many arrows as required for the diagram. Multiple output boundary arrows that represent the same physical and data objects shall join to form a single output boundary arrow segment. Branching inputs and joining outputs are illustrated in Figure 32.

A boundary arrow segment shall connect an ICOM code to a box or a junction. A boundary arrow segment may not connect two ICOM codes.

9.3.1 ICOM coding of boundary arrows

An *ICOM code* maps a boundary arrow in a diagram to an arrow attached to an ancestral box of that diagram. The use of ICOM codes, although optional, is strongly encouraged because they are required to form reference expressions.

A *complete ICOM code* is a diagram feature reference that explicitly identifies a diagram, a box, a side of the box, and an ordinal point of attachment of an arrow to that side of that box in that diagram. Examples of complete ICOM codes are given in Table 4. ICOM codes are of two types: boundary ICOM codes and box ICOM codes. Boundary ICOM codes are used to map an untunneled boundary arrow in a child diagram to an arrow attached to the parent box that is detailed by that diagram. Box ICOM codes are used to map tunneled boundary arrows to an arrow attached to some ancestral box. Box ICOM codes and their use are discussed in 9.4.

9.3.2 Boundary ICOM code

When ICOM coding is used, the unconnected end of an untunneled boundary arrow shall be attached to a *boundary ICOM code*. A boundary ICOM code consists of an *ICOM letter* (I, C, O, or M) followed by an *ICOM number* (i.e., a nonzero positive integer). The ICOM letter signifies that the boundary arrow maps to an arrow attached to the parent box of the current diagram as an input, control, output, or mechanism. The ICOM number signifies the ordinal position at which that arrow is attached to the parent box. The order of attachment is counted from left to right for control and mechanism arrows or from top to bottom for input and output arrows. For example, the boundary ICOM code ***C3*** indicates that the untunneled boundary arrow attached to the ICOM code corresponds to the third control arrow attached to the top side of the decomposition diagram's parent box, counting from the left.

Boundary ICOM codes relate a child diagram to its own immediate parent box. Boundary ICOM codes are assigned anew for every decomposition diagram. The boundary ICOM codes of one diagram have no relationship to boundary ICOM codes on any other diagram. Boundary ICOM codes always map the boundary arrows of a child diagram to arrows attached to that diagram's own immediate parent box.

A boundary ICOM code indicates an arrow's role with respect to a child diagram's parent box. A parental output arrow requires an output boundary arrow in the child diagram. However, a parental arrow with another role—input, control, or mechanism—does *not* determine the role of its corresponding boundary arrow in the child diagram. The arrow roles of input, control, and mechanism may change between parent box and child diagram. Ordinarily, as shown in Figure 26, arrow roles for parent box and detailing diagram do match, i.e., an input for a parent box is also an input for one or more boxes in the child diagram. However, a control arrow attached to a parent box may be either input or control for boxes in the child diagram. Similarly, a mechanism for a parent box may be an input for one or more boxes in the child diagram. Figure 33 shows examples of boundary arrows that change their roles.

9.4 Tunneled arrows

A *tunneled arrow* hides physical and data objects from a reader's view as the arrow logically traverses one or more diagrams. An arrow may be tunneled when the arrow's meaning—the physical and data objects that the arrow represents—can be ignored for one or more diagrams that are hierarchically consecutive within a model.

An arrow may be tunneled for any diagram. An arrow may be tunneled for any number of hierarchically consecutive diagrams, including context diagrams and *leaf diagrams* without children. A tunneled arrow shall traverse at least one diagram before it may be made visible again. An arrow tunneled into or out of a box shall not be visible in the diagram that details that box; symmetrically, an arrow tunneled into or out of a diagram shall not be visible in the parent diagram.

Formally, a tunneled arrow may only exist between an ancestral box and a descendent diagram. Further, the most ancestral box of any tunneled arrow is box ***0*** of the ***A-0*** context diagram. A tunneled arrow may not disappear or reappear in the ***A-1*** context diagram nor in any higher context diagram.

9.4.1 Tunnel notation

The *tunnel notations* illustrated in Figures 34 and 35 mark the disappearance and reappearance of tunneled arrows. A tunnel notation shall be made by bracketing the head or the tail of an arrow segment within a pair of short, shallow arcs that are drawn to resemble a pair of left and right parentheses characters. Applying tunnel notation to an arrow is called "tunneling" the arrow.

There are two possible paired applications of tunnel notation. The first paired application of tunnel notation is used to tunnel input, control, and mechanism arrows *in*. The second paired application is used to tunnel output arrows *out*.

a) To tunnel an input, control, or mechanism arrow *in*, follow the steps below:

1) Tunnel notation shall be applied to the head of an input, control, or mechanism arrow where the arrow is attached to a box, as shown in Figure 34. Such a disappearing arrow is said to be "tunneled into the box." A model note which contains a diagram reference to the descendent diagram where the tunneled arrow reappears may be placed by the disappearing arrow.

2) The tunneled arrow shall be omitted from one or more hierarchically consecutive descendent diagrams. The tunneled arrow is now "in the tunnel." The tunneled arrow disappears until it reappears at the boundary of a descendent diagram.

3) Tunnel notation shall be applied to the tail of the input, control, or mechanism boundary arrow in the diagram where the tunneled arrow reappears, as shown in Figure 35. Such a reappearing arrow is said to be "tunneled into the diagram." If ICOM coding is used, the tail of this arrow shall be attached to a complete box ICOM code; if ICOM coding is not used, a model note that contains a complete box ICOM code for the ancestral box where the tunneled arrow disappears shall be placed by the reappearing arrow.

An arrow tunneled into a box shall also be tunneled into a descendent diagram that details that box. Conversely, an arrow tunneled into a diagram shall also be tunneled into an ancestral box of that diagram.

In the case that a required descendent diagram does not exist because the ***A0*** activity has not been decomposed to that level in a model, a model note shall be placed by the tunneled arrow where it tunnels into or out of a box. This model note shall identify as clearly as possible the activity that would be detailed by the diagram where the tunneled arrow presumably would reappear (if tunneled into the box) or disappear (if tunneled out of the box).

b) To tunnel an output arrow *out*, follow the steps below:

1) Tunnel notation shall be applied to the head of an output boundary arrow, as shown in Figure 35. Such a disappearing arrow is said to be "tunneled out of the diagram." If ICOM coding is used, the head of the arrow shall be attached to a complete box ICOM code; if ICOM coding is not used, a model note that contains a complete box ICOM code for the ancestral box where the tunneled arrow reappears shall be placed by the arrow.

2) The tunneled arrow shall be omitted from one or more hierarchically consecutive ancestral diagrams. The tunneled arrow is now "in the tunnel." The tunneled arrow disappears until it reappears attached to the side of an ancestral box.

3) Tunnel notation shall applied to the attached tail of the output arrow, as shown in Figure 34, in the diagram where the tunneled arrow reappears. Such a reappearing arrow is said to be "tunneled out of the box." A model note that contains a diagram reference for the descendent diagram where the tunneled arrow disappears shall be placed by the reappearing arrow.

An arrow tunneled out of a descendent diagram shall also be tunneled out of one of that diagram's ancestral boxes. Conversely, an arrow tunneled out of an ancestral box shall also be tunneled out of at least one of its descendent detailing diagrams.

9.4.2 Complete box ICOM code

A boundary arrow that is tunneled into or out of a diagram corresponds to an arrow attached to a box in an ancestral diagram. A complete box ICOM code is a reference expression that identifies the point of disappearance of a tunneled input, control, or mechanism arrow or the point of reappearance of a tunneled output boundary arrow. Just as an untunneled boundary arrow is mapped to an arrow attached to a diagram's immediate parent box on its parent diagram by a boundary ICOM code, a complete box ICOM code maps a tunneled boundary arrow to an arrow attached to a specific box in a specific ancestral diagram.

If ICOM coding is used, the unconnected end of a tunneled boundary arrow shall be attached to a *complete box ICOM code*. Following the syntax of a ***diagram.feature*** reference (see Clause 11), a complete box ICOM code is a *diagram.icom* reference, where the *icom* reference is a *box ICOM code*. A complete box ICOM code consists of a *diagram number*, a period, a *box number* (i.e., a single digit), an *ICOM letter* (**I**, **C**, **O**, or **M**), and an *ICOM number* (i.e., a nonzero positive integer). The diagram number specifies the ancestral diagram that contains the box specified by the ICOM box number. The ICOM letter signifies that the boundary arrow maps to an arrow attached to that ancestral box as an input, control, output, or mechanism. The ICOM number signifies the ordinal position at which that arrow is attached to the ancestral box. The order of attachment is counted from left to right for control and mechanism arrows or from top to bottom for input and output arrows. For example, the complete box ICOM code ***A42.3C3*** indicates that its attached tunneled boundary arrow corresponds to the third arrow, counting from the left, attached to the top or control side of the third box in diagram ***A42*** of the current model.

In the case that the ancestral box is box ***0*** of the ***A-0*** context diagram, the complete box ICOM code is ***A-0.0Ln***, where ***L*** is the ICOM letter and ***n*** is the ICOM number. Note that the diagram reference ***A-1*** is not allowed in an ICOM code. For example, the complete box ICOM code for the attachment of an arrow as the second control to box ***0*** in the ***A-0*** diagram would be ***A-0.0C2***. Because only one box ***0*** exists in an IDEF0 model (although it may be represented both in the ***A-0*** context diagram and in an ***A-1*** context diagram), the *box ICOM code* ***0C2*** is itself not ambiguous. Because of this, by syntactic convention, the context diagram reference may be omitted from the ICOM code for an arrow explicitly tunneled into box ***0*** of the context diagram.

Another important syntactic convention is allowed for the case that the ancestral box is box ***0*** of the ***A-0*** context diagram but the arrow is not explicitly shown in the context diagram. All external interfaces to any decomposition activity of the ***A0*** activity are logically required to be interfaced to the ***A0*** activity. However, because every decomposition activity is completely within the scope of the ***A0*** activity, it may not be graphically appropriate to explicitly render each such arrow or to express such arrows at an appropriate level of abstraction in the context diagram. In this case, the complete box ICOM code is simply ***A-0.0L***, where ***L*** is the ICOM letter and the shorthand allowed by syntactic convention is simply ***0L***. For example, a low-level output arrow, tunneled out of a diagram deep within a decomposition hierarchy, which the modeler chooses not to present in the context diagram, may use ***0O*** as its complete box ICOM code.

Figures 36 through 41 illustrate the correct representation of tunneled arrows.

9.5 Model notes

A model note is a block of text or a graphical figure, such as an icon or photograph, that has been placed *in* a diagram by the diagram's author. Model notes are optional elements of a diagram and have no syntactic significance. However, once placed by an author, a model note is considered a permanent and essential component of a diagram. (In contrast, *reader notes* may be placed *on* a diagram by readers and authors when a model is reviewed. Reader notes are transient comments, questions, and responses *about* a diagram but are not part of the diagram.) In a diagram, each model note shall be identified by an integer number placed inside a small square; such a boxed number is called a *model note number*. Numerous examples of model note numbers and the model notes they identify are in the figures that illustrate this document.

In a given diagram, model note numbers shall form a consecutive sequence, starting with the number **1**. A reference to a model note in accompanying text may use an alternate notation that is supported by

commonly available fonts: a model note number may be written as an integer number bracketed by vertical lines. For example, the notation **|4|** refers to model note **4** in a diagram[4].

A model note provides information that is an integral part of a diagram's message but is not well expressed by IDEF0 box-and-arrow syntax. If a model note refers to a specific diagram feature, this feature shall be identified by an diagram feature reference in the model note. If a model note applies to more than one place or feature of a diagram, a copy of its model note number may be placed by each of these features. A model note number shall be in the same diagram as the model note it identifies.

10. IDEF0 reference expressions

The basic unit of an IDEF0 reference is the node number, which identifies a function in an IDEF0 model. The IDEF0 language defines the origin of the IDEF0 coordinate system to be the node number ***A0*** and provides a carefully designed scheme of expressions based upon node numbers that allows every diagram, every diagram feature, and every diagram page to be unambiguously identified in an IDEF0 model.

The first part of a node number is normally a diagram number and the last part of a node number is always a box number. A typical node number is ***A435***; this node number identifies the function represented by box ***5*** in diagram ***A43***. In turn, a diagram number is generally the node number of the function that is detailed by a diagram. The diagram number ***A43*** indicates that this diagram presents the decomposition of function ***A43***. The node number ***A43*** indicates the function represented by box ***3*** in diagram ***A4***, and in turn diagram ***A4*** presents the decomposition of function ***A4***.

a) *Predefined diagram numbers.* However, there are certain context diagrams in an IDEF0 model that do not have a parent function. Due to the recursive way in which node numbers are defined in terms of diagram numbers and diagram numbers are defined in terms of node numbers, the IDEF0 language provides predefined diagram numbers for these cases. These predefined diagram numbers allow node numbers to be defined for functions represented by boxes in diagrams that have no parent function. These predefined diagram numbers, as well as predefined node numbers and box numbers, are specified in Table 1.

   1) *The predefined diagram number* ***A-0***. The most important predefined diagram number is the ***A-0*** diagram number that identifies the required ***A-0*** context diagram. The IDEF0 language defines the ***A0*** function to be the single function represented in this diagram. However, no function in an IDEF0 model may be identified by the node number ***A-0***.

   2) *Other predefined context diagram numbers.* Other optional context diagrams may also be provided for a model. A model may contain from one to nine such context diagrams, providing the environmental context for the ***A0*** function and similar contexts for as many as eight lineal ancestors of the ***A0*** function. Because these context diagrams are optional, any of the nine possible diagrams may be the highest diagram in a model. Just as the ***A-0*** diagram has no parent function, the highest optional context diagram will have no parent function. Because any of these nine possible context diagrams may be the highest context diagram in a model and thus any of these nine possible context diagrams may have no parent function that can be used as a diagram number, the IDEF0 language provides predefined diagram numbers for all nine cases. Thus, in addition to the predefined diagram number ***A-0***, the other predefined diagram numbers provided by the IDEF0 language are: ***A-1***, ***A-2***, ***A-3***, ***A-4***, ***A-5***, ***A-6***, ***A-7***, ***A-***

[4]This use of straight lines contrasts to *reader notes*, which are written with parentheses. The roundness of the parentheses visually alludes to the circle that designates a *reader note number* on a model page. Similarly, the straightness of the model note's bracketing vertical bars visually alludes to the square that designates a model note number in a diagram. For example, a model reviewer might write, "On this diagram, reader note **(2)** questions the sufficiency of the claim made by model note **|5|** for the error management provided by the transform ***A3543*** as modeled by box ***3***.

8, and ***A-9***. These predefined diagram numbers may be referred to as "negative diagram numbers."

b) *Predefined node numbers.* Due the recursive definition of diagram numbers and node numbers, the predefined diagram numbers for optional context diagrams are coupled with similarly predefined node numbers. These predefined node numbers allow the diagram numbers for context diagrams lower than the highest context diagram to follow the rule that a diagram number is the same as the node number of the function that is parent to the diagram. The IDEF0 language provides these predefined node numbers for functions in context diagrams: ***A-8***, ***A-7***, ***A-6***, ***A-5***, ***A-4***, ***A-3***, ***A-2***, ***A-1***, and ***A0***. These predefined node numbers, including node number ***A0***, may be referred to as "negative node numbers."

c) *Predefined box numbers.* Because a node number is defined to include a box number, the predefined node numbers for functions in optional context diagrams are coupled with similarly predefined box numbers. These predefined box numbers allow the predefined node numbers to follow the rule that the last part of a node number is always a box number. The IDEF0 language provides these predefined box numbers for boxes in context diagrams: ***0***, ***-1***, ***-2***, ***-3***, ***-4***, ***-5***, ***-6***, ***-7***, and ***-8***. These predefined box numbers, including box number ***0***, may be referred to as "negative box numbers."

d) *Negative references.* The hyphen character in negative box numbers, negative node numbers, and negative diagram numbers shall be read as the word "minus." For example, the box number "-8" is pronounced "minus eight," the node number "A-13" is pronounced "A minus one three," the diagram number "A-0" is pronounced "A minus zero," and the diagram number "A-924" is pronounced "A minus nine two four."

e) *Parenting the **A0** diagram.* The box number ***0*** appears in both the ***A-0*** context diagram and the ***A-1*** context diagram to identify the box that represents the ***A0*** function of a model. Box ***0*** is defined by the IDEF0 language to represent the function identified by the node number ***A0*** and to be the parent box of the ***A0*** diagram.

Table 1—Predefined diagram numbers, node numbers, and box numbers

Predefined diagram number[a]	Predefined node number[b, c]	Predefined box number[d]	Node numbers of possible functions in diagram[e, f] and example node number sequence in diagram	Example box number sequence in diagram[g]	Comment
A-9[h]	A-8	-8	**A-8**, A-91, A-92, A-93, …, A-99 A-91, A-92, A-8, A-94, A-95	 1, 2, -8, 4, 5	Highest possible context diagram. Optional context diagram. Defined as parent of A-8 diagram.
A-8	A-7	-7	**A-7**, A-81, A-82, A-83, …, A-89 A-81, A-7, A-83, A-84, A-85	 1, -7, 3, 4, 5	Optional context diagram. Defined as parent of A-7 diagram.
A-7	A-6	-6	**A-6**, A-71, A-72, A-73, …, A-79 A-71, A-72, A-73, A-74, A-6	 1, 2, 3, 4, -6	Optional context diagram. Defined as parent of A-6 diagram.
A-6	A-5	-5	**A-5**, A-61, A-62, A-63, …, A-69 A-61, A-62, A-5, A-64, A-65	 1, 2, -5, 4, 5	Optional context diagram. Defined as parent of A-5 diagram.
A-5	A-4	-4	**A-4**, A-51, A-52, A-53, …, A-59 A-4, A-52, A-53, A-54, A-55	 -4, 2, 3, 4, 5	Optional context diagram. Defined as parent of A-4 diagram.
A-4	A-3	-3	**A-3**, A-41, A-42, A-43, …, A-49 A-41, A-42, A-3, A-44, A-45	 1, 2, -3, 4, 5	Optional context diagram. Defined as parent of A-3 diagram.
A-3	A-2	-2	**A-2**, A-31, A-32, A-33, …, A-39 A-31, A-32, A-33, A-2, A-35	 1, 2, 3, -2, 5	Optional context diagram. Defined as parent of A-2 diagram.
A-2	A-1	-1	**A-1**, A-21, A-22, A-23, …, A-29 A-21, A-22, A-23, A-24, A-1	 1, 2, 3, 4, -1	Optional context diagram. Defined as parent of A-1 diagram.
A-1	A0	0	**A0**, A-11, A-12, A-13, …, A-19 A-11, A0, A-13, A-14, A-15	 1, 0, 3, 4, 5	Optional context diagram. Defined as parent of A0 diagram.
A-0	A0	0	**A0** A0	 0	Required context diagram. Defined as primary parent of A0 diagram.
A0	A1, A2, A3 A4, A5, A6 A7, A8, A9[i]	1, 2, 3 4, 5, 6 7, 8, 9	**A1, A2**, A3, …, A9 A1, A2, A3, A4, A5	 1, 2, 3, 4, 5	Required decomposition diagram.

[a] A diagram identified by a predefined diagram number shall contain a box to model the function whose node number is given in the second column. This box shall have the box number given in the third column. For example, diagram A-2 will contain a box with the box number -1 to represent function A-1.

[b] Each predefined node number is associated with the specific diagram whose diagram number is given in the first column. For example, function A-4 may only be represented by a box in diagram A-5.

[c] Each predefined node number identifies a function that is defined as the parent of the diagram with the predefined diagram number in the first column of the next row. The only exception to this pattern is function A0 for the A-1 diagram, which is defined as a parent of the A0 diagram. The A-1 diagram is not a parent of the A-0 diagram, and the A0 function is not a parent of the A-0 diagram. For example, function A-7 is defined as the parent of diagram A-7, and conversely, the context diagram A-7 is defined as the decomposition of function A-7.

[d] The box for a function with predefined node number shall be found in the specific diagram whose diagram number is given in the first column. For example, the box number -2 will be used only in diagram A-3.

[e] A node numbers in bold type indicates a function that must be modeled by a box in the diagram identified by the diagram number in the first column.

[f] A minimum of two functions must be modeled by boxes in a diagram (see 9.1). For optional context diagrams, one of these functions shall be the function with the highlighted node number. See the column *Example Box Numbers in Diagram* for examples of these required substitutions.

[g] For the purposes of these examples, a typical diagram is assumed to contain 5 boxes. The box to represent the required function in these examples has been chosen randomly.

[h] The A-9 function implied by diagram number A-9 is never actually modeled by a box in an IDEF0 model.

[i] By strict construction, these node numbers should be: A01, A02, A03, A04, A05, A06, A07, A08, A09. However, the zero is omitted by IDEF0 conventions.

10.1 Box numbers

A *box number* (see 5.1) is an expression that uniquely identifies a box within a given diagram. Every box in a diagram is assigned a box number that is unique within that diagram. The IDEF0 box numbering scheme is designed to allow recursive construction of node numbers by concatenating a box number to a diagram number. These box numbers are also used to cross-reference information in accompanying model pages to boxes in a diagram.

a) *A-0 context diagram.* The box number ***0*** is assigned by the IDEF0 language to any box that represents the ***A0*** function of a model. Thus, the box number ***0*** shall be assigned to the single box representing the function ***A0*** in the ***A-0*** context diagram.

b) *Decomposition diagrams.* Boxes in a decomposition diagram shall be sequentially assigned box numbers from the series of integer numbers ***1*** through ***9***. Since boxes are normally arranged diagonally from the top left corner to the bottom right corner of a diagram, box numbers shall be assigned in sequence, starting with box number ***1*** for the box at the top left of a diagram. If off-diagonal boxes are also used, the numbering sequence shall start with the on-diagonal boxes and then continue, from the lower right, in counter-clockwise order. For example, should there be four boxes in a decomposition diagram, counting from the diagram's upper left corner, the box numbers of these boxes would form the following sequence: ***1***, ***2***, ***3***, ***4***.

c) *Optional context diagrams.* A box that represents the ***A0*** function or one of its lineal ancestors in an optional context diagram may be any box in that diagram. As with decomposition diagrams, box numbers in an optional context diagram shall be assigned in sequence, starting with box number ***1*** for the box at the top left of the diagram. However, as specified in Table 1, the appropriate predefined negative box number shall replace the box number of the box that represents the ***A0*** function or its ancestor. The other boxes in an optional context diagram retain their expected box numbers. For example, should an ancestor to the ***A0*** function be represented by the third box of four in an ***A-6*** context diagram, counting from the diagram's upper left corner, the box numbers assigned to these boxes would form the following sequence: ***1***, ***2***, ***-5***, ***4***.

10.2 Node numbers

A *node number* is an expression that uniquely identifies a function and its position in a model hierarchy. A node number is normally formed by concatenating a diagram number and a box number. The diagram number identifies the diagram in which the function is represented as a box. The box number identifies that box in the diagram. See Table 1 for the definition of exceptional cases.

a) *Node letter*. The first character of a node number is a node letter. (This node letter distinguishes node numbers from box numbers in the ***A-0*** and ***A0*** diagrams.) Conventionally, the node letter used in IDEF0 models is an uppercase "**A**". However, a node letter may be any uppercase alphabetic character. Within a single model, the selected node letter shall be used for all node numbers. In a set of related models, different node letters may be used to construct the node numbers of different models. Alternative decompositions invoked by a call arrow are considered different submodels and, as such, the node numbers of each submodel may begin with a different node letter. (Because a diagram number is defined in terms of node numbers, a diagram number begins with the same node letter that is used to construct node numbers for a model.)

b) *A-0 context diagram.* The node number ***A0*** is assigned by the IDEF0 language to the function represented by box ***0*** in the ***A-0*** context diagram.

c) *Decomposition diagrams.* The node number for a function in a decomposition diagram shall be constructed by appending a box number to a diagram number. The box number shall identify the box that represents the function. This diagram number shall identify the decomposition diagram that contains this box. For example, should a function be detailed by five boxes in diagram ***A19***, the node number of the function represented by box ***2*** in this decomposition diagram will be ***A192***.

Strictly following this construction of node numbers, the functions represented by the boxes in the ***A0*** diagram would be given the node numbers ***A01***, ***A02***, ***A03***, …, ***A09***. However, by IDEF0 convention, the leading numeral zero is always omitted from the node numbers of these functions. Thus, the node numbers of functions in the ***A0*** diagram shall be ***A1***, ***A2***, ***A3***, …, ***A9***.

d) *Optional context diagrams.* For a function other than the ***A0*** function or one of its lineal ancestors, the node number for a function in an optional context diagram shall be constructed by appending a box number to a diagram number, just as a node number is determined for a function in a decomposition diagram. The box number shall identify the box that represents the function. The diagram number shall identify the context diagram that contains this box. However, as specified in Table 1, the appropriate negative node numbers shall be assigned to the ***A0*** function and its lineal ancestors, i.e., to those functions represented by boxes with negative box numbers in optional context diagrams. For example, the grandfather of the ***A0*** function will be assigned the node number ***A-2***. Should this grandfather function be represented by the second box in diagram ***A-3***, the node number of the function represented by the first box in this diagram will be ***A-31***.

10.3 Diagram numbers

A *diagram number* is an expression that uniquely identifies a diagram and its hierarchic position within an IDEF0 model. If a diagram has a parent function, the node number of the diagram's parent function shall be assigned as the diagram number of the diagram.

Certain context diagrams in an IDEF0 model that do not have a parent function and thus have no corresponding parent node number. The IDEF0 language provides diagram numbers for these diagrams. These predefined diagram numbers are specified in Table 1.

a) *A-0 context diagram.* The diagram number ***A-0*** is assigned by the IDEF0 language to the required context diagram that contains only one box, box ***0***, to represent a model's ***A0*** function. This number is the only diagram number in an IDEF0 model that may not be used to construct node numbers.

b) *Decomposition diagrams.* The node number of a decomposition diagram's parent function shall be assigned as the diagram number for the diagram. For example, the diagram number of the decomposition diagram that details function ***A47*** is simply ***A47***.

c) *Optional context diagrams.* For an optional context diagram that has a parent function in a parent context diagram, the node number of this parent function shall be assigned as the diagram number for the diagram, just as a diagram number is determined for a decomposition diagram. However, an optional context diagram that does not have a parent function shall be assigned the appropriate predefined negative diagram number as specified in Table 1. For example, an optional context diagram that details some function ***A-15***, represented by box ***5*** on diagram ***A-1***, would be assigned the diagram number ***A-15***. However, the diagram number ***A-1*** is assigned to the optional context diagram that contains box ***0*** as specified in Table 1.

d) *Diagram numbers vs. node numbers.* A function is identified by its node number while a diagram is identified by its diagram number. While diagram numbers and node numbers are designed to look alike and while the set of node numbers and the set of diagram numbers in a model largely overlap, diagram numbers and node numbers differ in critical ways. First, diagram numbers include the predefined diagram numbers for context diagrams that do not have parent functions. Second, node numbers include node numbers constructed for leaf nodes, which do not have decomposition diagrams. As a result, the set of node numbers will be from two to nine times larger than the set of diagram numbers for a given IDEF0 model.

Table 2 illustrates the diagram numbers, node numbers, and box numbers that may be found in a typical IDEF0 model.

Table 2—Diagram, function, and box reference examples

Diagram numbers: diagrams	Box numbers: boxes	Node numbers: functions	Comments
			Optional high-level context diagrams
A-93	1, 2, …, 9	A-931, A-932, …, A-939	Optional context diagram
A-4134	1, 2, …, 9	A-41341, A-41342, …, A-41349	Optional context diagram
A-2	1, -1, 3, …, 9	A-21, A-1, A-23, …, A-29	Optional context diagram
A-26	1, 2, …, 9	A-261, A-262, …, A-269	Optional context diagram
A-1	1, 2, 0, 4, …, 9	A-11, A-12, A0, A-14, …, A-19	Optional context diagram (contains box *0* representing the *A0* function)
A-14	1, 2, …, 9	A-141, A-142, …, A-149	Optional environmental context diagram
A-0	0	A0	Required context diagram (contains box *0* representing the *A0* function)
A0	1, 2, …, 9	A1, A2, …, A9	Required decomposition diagram
A1, A2, …, A9	1, 2, …, 9	A11, A12, …, A19, …, A91, A92, …, A99	Possible decomposition diagrams
A1	1, 2, …, 9	A11, A12, …, A19	Decomposition diagram
A5	1, 2, …, 9	A51, A52, …, A59	Decomposition diagram
A11, A12, …, A19, …, A91, A92, …, A99	1, 2, …, 9	A111, A112, …, A199, …, A911, A912, …, A999	Possible decomposition diagrams
A32	1, 2, …, 9	A321, A322, …, A329	Decomposition diagram
A49	1, 2, …, 9	A491, A492, …, A499	Decomposition diagram
A61	1, 2, …, 9	A611, A612, …, A619	Decomposition diagram
A63	1, 2, …, 9	A631, A632, …, A639	Decomposition diagram
A87	1, 2, …, 9	A871, A872, …, A879	Decomposition diagram
A111, A112, …, A199, …, A911, A912, …, A999	1, 2, …, 9	A1111, A1112, …, A1999, …, A9111, A9112, …, A9999	Possible decomposition diagrams
A125	1, 2, …, 9	A1251, A1252, …, A1259	Decomposition diagram
A153	1, 2, …, 9	A1531, A1532, …, A1539	Decomposition diagram
A211	1, 2, …, 9	A2111, A2112, …, A2119	Decomposition diagram
A639	1, 2, …, 9	A6391, A6392, …, A6399	Decomposition diagram
A762	1, 2, …, 9	A7621, A7622, …, A7629	Decomposition diagram
			Possible further decomposition diagrams
A1111	1, 2, …, 9	A11111, A11112, …, A11119	Decomposition diagram
A2451	1, 2, …, 9	A24511, A24512, …, A24519	Decomposition diagram
A46223	1, 2, …, 9	A462231, A462232, …, A462239	Decomposition diagram
A999999	1, 2, …, 9	A9999991, A9999992, …, A9999999	Decomposition diagram

10.4 Node tree

The developed IDEF0 model with its structured decomposition provides the basis to describe the full decomposition as a *node tree* on a single FEO page. A node tree is not required for a valid IDEF0 model. The content of a node tree shall be identical to that of a node index prepared for the same model (see 10.5).

There is no standard format for the display of node information in a node tree, except that the model hierarchy shall be shown graphically as a tree rooted at a chosen node (e.g., the ***A0*** function for a whole model). Figure 42 illustrates such a tree.

10.5 Node index

The *node index* presents a node tree in the format of a document outline. Each node number, with either its corresponding diagram title or function name, shall be presented in an indented form that exhibits the nested hierarchic structure of the model. This places related diagrams together in an order often used to create an ordinary Table of Contents, as is illustrated in Figure 43.

10.6 Diagram references

A *diagram reference* is an expression that uniquely identifies a diagram and its position in the model hierarchy of a specific model. A diagram reference shall be constructed by putting a model name abbreviation and a slash ("/") character in front of a diagram number. For example, the diagram reference for the ***A312*** diagram in a model known by the model name abbreviation ***QA*** would be ***QA/A312***. References to a diagram in the same model may omit the model name abbreviation, using only the diagram number.

10.7 Page references

A *page reference* is an expression that uniquely identifies each model page in an IDEF0 model and links each model page to a specific IDEF0 diagram. A page reference shall be constructed by appending *page type letters* and, as required, sequence numbers to a diagram reference. A page type letter is an uppercase letter that denotes a specific type of model page. The page type letters "**D**" for diagram, "**F**" for FEO, "**T**" for text, "**G**" for glossary, and "**K**" for kit shall be used to denote these defined types of model pages. Other page type letters may be defined with meanings and usage specific to given models and modeling projects.

If there is more than one model page of a specific type associated with a given diagram or other model page, the model pages within that type shall be distinguished by a sequence number following the page type letter. For example, the page reference for a single text page associated with the diagram ***MDL/A33*** might be ***MDL/A33T***. The page reference for this same text page after additional text pages have been added to discuss diagram ***A33*** would become ***MDL/A33T1***. The page reference for the third FEO page associated with diagram ***A321*** in model ***QA*** would be ***QA/A321F3***, where ***QA/A321*** is the diagram reference, ***F*** is the page type letter, and ***3*** is the sequence number.

If a page reference does not show an explicit sequence number, the sequence number is assumed to be ***1***. If a page reference does not show an explicit page type letter, the page type letter is assumed to be ***D*** and the model page is assumed to be a diagram page. Thus, a diagram reference and the page reference for that diagram are consistent: ***MJN/A473D1*** and ***MJN/A473*** are equivalent page references.

Except for a diagram page, in general, any type of model page may supplement any other type of model page. For example, given diagram ***A42*** in model ***QRK***, there may be a glossary page that defines terms used in the second text page that explains the third FEO page that supplements this diagram. The fully expanded page reference for this glossary page would be ***QRK/A42D1F3T2G1***. The slightly shorter ***QRK/A42F3T2G*** is an equivalent page reference.

11. IDEF0 diagram feature references

A standard notation shall be used in writing to refer to specific parts of diagrams. These *diagram feature references* are based on box numbers, ICOM codes, and note numbers. Diagram feature references may be combined with diagram references using "dot notation," that is, by concatenating a diagram-based reference, a period (the "dot"), and a diagram feature reference. Such **diagram.feature** (read "diagram-see-feature") references are illustrated in Tables 3 through 6.

a) *Box references.* An activity box is referenced by its box number. Dot notation may join a box number to a diagram reference to make a *diagram.box* reference (see Table 3).

Table 3—Reference notation for box references

Reference	Example	Interpretation
box number	3	Specifies a box in the current diagram of the current model. The example *3* refers to the box with box number 3. This example may be read as "box 3" or "the activity modeled by box 3." Often italicized in text; for example, "The name in *3* should not differ in font size from other box names in the diagram," which refers to a box, or "The name of *3* should reflect the activity's purpose rather than the inputs it consumes," which refers to a function.
diagram.box	A42.3	Specifies a box in a specific diagram of the current model. The example *A42.3* refers to the box with box number 3 in diagram A42 of the current model. This example may be read as "in diagram A42, see box number 3" or as "see activity A423."
diagram.box	MFG/A42.3	Specifies a box in a specific diagram of a specific model. The example *MFG/A42.3* refers to the box with box number 3 in diagram A42 of the model MFG. This example may be read as "see box 3 in diagram A42 of the model abbreviated MFG" or as "in the Manufacturing model, see function A423."

b) *ICOM codes (box/arrow references).* A box/arrow reference is either a *boundary ICOM code* or a *box ICOM code*. Dot notation may join either form of ICOM code to a diagram reference to form a *diagram.icom* reference. A *diagram.icom* reference is also known as a *complete ICOM code* (see Table 4).

Table 4—Reference notation for ICOM codes

Reference	Example	Interpretation
boundary ICOM code	I2	Strictly, specifies the ordinal attachment of some arrow to a specific side of the parent box that is detailed by the current diagram of the current model. Interpreted strictly, the example *I2* refers to the second arrow attached to the input side of the box that is detailed by the current diagram of the current model. This example may be read as "the second input of the current function." Loosely, identifies the boundary arrow that maps to an arrow attached, at the specified ordinal position, to a specific side of the box that is detailed by the current diagram of the current model. Interpreted loosely, the example *I2* refers to the boundary arrow that maps to the second input arrow attached to the box that is parent to the current diagram of the current model. This example may be read as "the boundary arrow I2." A boundary ICOM code is a *complete ICOM code* that assumes that the unspecified box is the parent box detailed by the current diagram, that the unspecified diagram is the current diagram's parent diagram, and that the unspecified model is the current model.

Table 4—Reference notation for ICOM codes *(Continued)*

Reference	Example	Interpretation
diagram.icom reference or *complete ICOM code*	A42.I2	Specifies the ordinal attachment of some arrow to a specific side of the parent box that is detailed by a specific diagram of the current model. The example *A42.I2* refers to the second arrow attached to the input side of diagram A42's parent box. This example may be read as "the second input of the function detailed in diagram A42," or simply as "in diagram A42, see the boundary arrow I2."
diagram.icom reference or *complete ICOM code*	MFG/A42.I2	Specifies the ordinal attachment of some arrow to a specific side of the parent box that is detailed by a specific diagram of a specific model. The example *A42.I2* refers to the second arrow attached to the input side of diagram A42's parent box in the model with the abbreviated name MFG. This example may be read as "the second input of the function detailed in diagram A42 of model MFG," or simply as "in diagram A42 of MFG, see the boundary arrow I2."
box ICOM code	3I2	Strictly, specifies the ordinal attachment of some arrow to a specific side of a specific box in the current diagram of the current model. Interpreted strictly, the example *3I2* refers to the attachment of the second arrow, counting from the top, to the left or input side of box 3 in the current diagram of the current model. This example may be read as "box 3, input 2." Loosely, identifies the arrow that is attached, at the specified ordinal position, to a specific side of a specific box in the current diagram of the current model. Interpreted loosely, the example *3I2* refers to the second input arrow attached to box 3 in the current diagram of the current model. This example may be read as "the arrow attached at 3I2." A box ICOM code is a *complete ICOM code* that assumes that the unspecified diagram is the current diagram and that the unspecified model is the current model.
diagram.icom reference or *complete ICOM code*	A42.3I2	Specifies the ordinal attachment of an arrow to a specific side of a specific box in a specific diagram of the current model. The example *A42.3I2* refers to the second arrow attached to the left or input side of box 3 in diagram A42 of the current model. This example may be read as "in diagram A42, see the second input to box number 3."
diagram.icom reference or *complete ICOM code*	MFG/A42.3I2	Specifies the ordinal attachment of an arrow to a specific side of a specific box in a specific diagram of a specific model. The example *MFG/A42.3I2* refers to the second arrow attached to the left or input side of the box with box number 3 in diagram A42 of the model abbreviated MFG. This example may be read as "see box 3, input 2, in diagram A42 in model MFG."

c) *Arrow references.* An arrow reference consists of one or more ICOM codes that together unambiguously identify an arrow, an arrow segment, or an arrow path. Dot notation may join an arrow reference to a diagram reference to form a *diagram.arrow* reference (see Table 5).

Table 5—Reference notation for arrow references

Reference	Example	Interpretation
arrow reference	3I2	See Table 4. As an arrow reference, the example *3I2* refers to the arrow segment attached as specified by the box ICOM code I2 in the current diagram of the current model. This example may be read as “the arrow attached at 3I2” or “the second input to box 3.”
arrow reference	2O3 to 3I2 *or* 2O3:3I2	Specifies the internal source and the internal use of an arrow in the current diagram of the current model. The example *2O3 to 3I2* refers to the arrow that connects the third output of box 2 to the second input of box 3. This example may be read as “the branch of the third output arrow of box number 2 that provides the second input for box 3.”
diagram.arrow reference	A42.2O3 to 3I2 *or* A42.2O3:3I2	Specifies the source and the use of an arrow in a specific diagram of the current model. The example *A42.2O3 to 3I2* refers to the arrow that connects the third output of box 2 to the second input of box 3 in diagram A42 of the current model. This example may be read as “in diagram A42, see the arrow that starts as the third output of box 2 and ends as the second input of box 3.”
arrow reference	I2	See Table 4. As an arrow reference, the example *I2* refers to the boundary arrow attached to boundary ICOM code I2 *and all its branches* in the current diagram of the current model. This example may be read as “the boundary arrow I2.”
arrow reference	I2 to 3I2 *or* I2:3I2	Specifies the external source and the internal use of an arrow in the current diagram of the current model. The example *I2:3I2* refers to the arrow that connects the second input of this diagram’s parent box to the second input of box 3 in the current diagram of the current model. This example may be read as “the branch of I2 that is the second input of box 3.”
arrow reference	3O1 to O3 *or* 3O1:O3	Specifies the internal source and the external use of an arrow in the current diagram of the current model. The example *3O1 to O3* refers to the arrow in the current diagram of the current model that connects the first output of box 3 to the third output of this diagram’s parent box. This example may be read as “the first output arrow of box 3, which becomes the third output of this diagram’s parent box.”
INVALID REFERENCE	I2 to O3 *nor* I2:O3	*Each boundary input arrow shall be transformed by some function in that diagram. Each boundary output arrow shall be the product of a transformation by some function in that diagram. No arrow may traverse a diagram directly from one boundary ICOM to another boundary ICOM.*
arrow path reference	I2 to 3I2 to 3O1 to (4C1 and 5C2) or I2:3I2:3O1 :(4C1,5C2)	Specifies a sequence of arrows from an initial source to a final use through intermediate uses and sources. The example refers to three arrows (I2 to 3I2, 3O1 to 4C1, and 3O1 to 5C2) that trace a path from the initial source (I2) to two final uses (4C1 and 5C2). Box 3 is the intermediate use (3I2) and source (3O1). This example may be read as “from the boundary arrow with ICOM code I2 to box 3, input 2, through the activation of box 3 that yields output 1, to the availability via branching of that output as control 1 on box 4 and control 2 on box 5.

d) *Note references*. A note reference is either a *model note number* or a *reader note number*. Dot notation may join either note number to a diagram reference to form a *diagram.note* reference (see Table 6a and Table 6b).

Table 6a—Reference notation for model note references (normative)

Reference	Example	Interpretation
model note number	3	Specifies a model note on the current diagram of the current model. The example 3 refers to the model note with the model note number 3 in the current diagram of the current model. This example may be read as "model note 3." This notation for a model note number *shall* be used in a diagram to designate a model note. This notation may also be used in reference expressions.
model note number (reference-only notation)	|3|	This alternate notation may be used only in reference expressions. This notation *shall not* be used in a diagram as a model note number. The example *|3|* refers to model note 3, just as the model note reference of the previous example.
diagram.note reference	A42. 3	Specifies a model note in a specific diagram of the current model. The example *A42.* 3 refers to the model note with the model note number 3 in diagram A42 of the current model. This example may be read as "in diagram A42, see model note number 3."
diagram.note reference	MFG/A42.|3|	Specifies a model note in a specific diagram of a specific model. The example *MFG/A42.|3|* refers to the model note with the model note number 3 in diagram A42 of the model MFG. This example may be read as "see the third model note in diagram A42 of the model MFG."

Table 6b—Reference notation for reader note references (informative)

Reference	Example	Interpretation
reader note number	③	Specifies a reader note on the current diagram of the current model. The example ③ refers to the reader note with the reader note number 3 on the current diagram of the current model. This example may be read as "reader note 3." This notation for a reader note number *shall* be used on a diagram to designate a reader note. This notation may also be used in reference expressions.
reader note number (reference-only notation)	(3)	This alternate notation may be used only in reference expressions. This notation *shall not* be used on a diagram as a reader note number.
diagram.note reference	A42.(3)	Specifies a reader note on a specific diagram of the current model. The example *A42.(3)* refers to reader note 3 on diagram A42 of the current model. This example may be read as "see reader note 3 on diagram A42."
diagram.note reference	MFG/A42.③	Specifies a reader note on a specific diagram of a specific model. The example *MFG/A42.*③ refers to reader note 3 on diagram A42 of the model MFG. This example may be read as "on diagram A42 of the model MFG, see reader note number 3."

Annex A

(informative)

Bibliography

[B1] IEEE Std 1320.2-1998, Standard for Conceptual Modeling Language—Syntax and Semantics for IDEF1X$_{97}$ (*IDEF$_{object}$*).

[B2] FIPS PUB 183, *Integration Definition for Function Modeling (IDEF0)*, National Institute for Standards and Technology, December 1993.

[B3] FIPS PUB 184, *Integration Definition for Information Modeling (IDEF1X)*, National Institute for Standards and Technology, December 1993.

Annex B

(informative)

IDEF0 language: abstract formalization

B.1 IDEF0 abstract syntax

This annex presents a mathematical formalization of the syntax of the IDEF0 function modeling method. A formal syntax determines exactly the class of well-formed constructs of the language and hence unambiguously answers the question of whether any given construction is syntactically legitimate. The abstract formalization itself consists chiefly of a series of mathematical definitions. However, the bulk of this annex consists of informal exposition to clarify and motivate the unadorned definitions.

This annex presents a formal specification of the syntax of the IDEF0 modeling language. The syntax is an *abstract* syntax, in the sense that it is presented at a high level of abstraction applicable to any particular form of representation for an IDEF0 model, whether the traditional graphical syntax or a "linear" equivalent in a logical language like the knowledge interchange format (KIF). This capacity in turn supports enterprise model integration, not only within the IDEF suite of methods, but also with regard to other modeling methods [e.g., entity relationship (ER), NIAM] and other naming conventions, both within and across enterprise contexts.

In general, an IDEF0 diagram can be thought of as a certain kind of mathematical object known as a *graph*. Intuitively, a graph is any collection of objects that are linked or connected together in some way. Objects so linked are said to be *adjacent*. It is not necessary that every object in the set be linked to some other object; any way of linking some of the objects in a given set to other (possibly the same) objects in the set is an acceptable graph. Pictorially, one can think of a (finite) graph as the result of starting with some set of dots on a sheet of paper and drawing lines between pairs of dots (no more than one line per pair of dots) or from a dot to itself (no more than one per dot). Formally, graphs are represented by listing these two aspects of a graph separately: the set of objects—more correctly known as its *vertices*—and the set of links—more correctly known as its *edges*.[5] Because the nature of an edge in an ordinary graph is irrelevant beyond the fact that it links the particular vertices that it does, and because there can be only one edge from one vertex to another in a graph, an edge between one vertex ***N1*** and another ***N2*** is usually represented simply as the unordered set ***{N1,N2}*** consisting of exactly those two vertices. (Graph-like structures that permit multiple edges between vertices are called *multigraphs*.) Two vertices ***N1*** and ***N2*** are thus adjacent in a graph if and only if the set ***{N1,N2}*** is an edge in the graph.

When the edges of a graph have an associated direction, so that one can speak of an edge extending from one vertex to another (as opposed to simply linking two vertices), the graph is known as a directed graph. (In terms of the pictorial analogy above, envision changing all the lines in the picture to arrows with one arrowhead.) Because the element of directionality adds a little more richness to the notion of an edge, it is now possible for there to be two edges between vertices; these edges must, however, "point" in different directions. That is, it is permissible to have an edge from ***N1*** to ***N2***, and from ***N2*** back to ***N1***. It is not, however, permissible to have more than one edge from ***N1*** to ***N2***. (Structures that permit more than one such edge are called *directed* multigraphs.)

[5]Vertices and edges are also commonly known as *nodes* and *arcs*, respectively. However, the term "node" has an entrenched meaning in IDEF0 and so it will not be used it here; thus, "arc" is also avoided because it is usually paired with "node."

Formally, the directedness of an edge is captured by *ordering* the pair of vertices the edge connects. Ordering is done simply by representing a directed edge from ***N1*** to ***N2*** as the *ordered pair* $\langle N1, N2 \rangle$, instead of the unordered set $\{N1, N2\}$. The direction of the edge is represented by the order of the vertices. Given this definition (since ordered pairs are identical if and only if (iff) they have the same first and second elements), it is clear that there can be no more than one edge from one vertex ***N1*** to another ***N2***.

In this formalization, an IDEF0 diagram (i.e., the content of a single diagram *page* in an IDEF0 model) is defined to be a special type of directed graph. A possible misconception must be avoided, however. Because it is common to represent directed edges in a graph as arrows, it is natural—initially at any rate—to think of an IDEF0 diagram as a graph whose vertices are the boxes of the diagram and whose edges are the arrows that connect them. For a number of good reasons—for instance, the complex ways in which arrows can branch and join in an IDEF0 diagram, and the fact that some arrows do not connect to any box at one end—this is not how IDEF0 diagrams are modeled in this formalization. Rather, in this formalization, the vertices of an IDEF0 diagram comprise not only boxes, but arrows—more exactly, the arrow *segments* that constitute them—and the junctions at which arrows branch and join as well. The edges of the diagram, rather than representing arrows, represent the points at which arrows segments connect to boxes and junctions. Because there are six types of connection (represented in standard IDEF0 diagrams by which side of a box an arrow segment connects to, and by branching and joining), edges come in six "flavors"—input, control, output, mechanism, junction-input, and junction-output.

Because IDEF0 diagrams are representations that carry information, they are naturally thought of as expressions in a *language*. Typically, a language has two components: a *lexicon* and a *grammar*. The lexicon comprises the basic building blocks of the language, and the grammar provides the rules by which complex representations are constructed from those building blocks. The lexicon is defined as follows:

Definition 1. IDEF0 lexicon

> An *IDEF0 lexicon* $\mathbf{L}$ is a triple $\langle \mathbf{B,S,J} \rangle$ such that $\mathbf{B}$, $\mathbf{S}$, and $\mathbf{J}$ are pairwise disjoint countable infinite sets. The elements of $\mathbf{B}$ are called *boxes*, the elements of $\mathbf{S}$ are called *arrow segments*, and the elements of $\mathbf{J}$ are called *junctions*.

Intuitively, $\mathbf{B}$ and $\mathbf{S}$ will be the formal counterparts of the boxes and arrow segments of an IDEF0 diagram. Unlike boxes and arrow segments, junctions are not specific modeling *objects* in a traditional IDEF0 diagram, but rather are simply points of contact between arrow segments. However, it proves to be very useful for the semantical purposes to represent junctions explicitly, as will be seen below, as they function semantically much like boxes. $\mathbf{B}$, $\mathbf{S}$, and $\mathbf{J}$ are stipulated to be pairwise disjoint[6] because, of course, boxes, arrow segments, and junctions are all different sorts of things. It is stipulated that all three sets are infinite so that there are enough basic elements to build arbitrarily large (but nonetheless finite) models.

The grammar is the more complex element of a language. The following is the key definition for the IDEF0 language:

Definition 2. IDEF0 graph

> Let $\mathbf{L}=\langle \mathbf{B,S,J} \rangle$ be an IDEF0 lexicon. An *IDEF0 graph* (*in* $\mathbf{L}$) is a finite directed graph $\mathbf{G}=\langle \mathbf{V_G, E_G} \rangle$ such that $\mathbf{V_G}=\bigcup\{\mathbf{B_G, S_G, J_G}\}$, where $\mathbf{B_G} \subseteq \mathbf{B}$, $\mathbf{S_G} \subseteq \mathbf{S}$, and $\mathbf{J_G} \subseteq \mathbf{J}$, and $\mathbf{E_G}= \bigcup\{\mathbf{I_G, C_G, O_G, M_G, L_G, R_G}\}$, where $\mathbf{I_G}$, $\mathbf{C_G}$, and $\mathbf{M_G}$ are pairwise disjoint subsets of $\mathbf{S_G} \times \mathbf{B_G}$, $\mathbf{O_G} \subseteq \mathbf{B_G} \times \mathbf{S_G}$, $\mathbf{L_G} \subseteq \mathbf{S_G} \times \mathbf{J_G}$, and $\mathbf{R_G} \subseteq \mathbf{J_G} \times \mathbf{S_G}$ satisfying these conditions:

[6] A set $\mathbf{M}$ of sets is pairwise disjoint if and only if any two distinct members of $\mathbf{M}$ are disjoint, i.e., have no members in common.

(1) $\mathbf{E_G}$ is functional on $\bigcup\{\mathbf{I_G},\mathbf{C_G},\mathbf{M_G},\mathbf{L_G}\}$, that is, if $\langle \mathbf{x},\mathbf{y}\rangle \in \mathbf{E_G}$ and $\langle \mathbf{x},\mathbf{z}\rangle \in \mathbf{E_G}$, then either $\mathbf{y}=\mathbf{z}$ or both $\langle \mathbf{x},\mathbf{y}\rangle,\langle \mathbf{x},\mathbf{z}\rangle \in \bigcup\{\mathbf{O_G},\mathbf{R_G}\}$;

(2) $\mathbf{E_G}^{-1}$ is functional on $\bigcup\{\mathbf{O_G},\mathbf{R_G}\}$, that is, if $\langle \mathbf{y},\mathbf{x}\rangle \in \mathbf{E_G}$ and $\langle \mathbf{z},\mathbf{x}\rangle \in \mathbf{E_G}$, then either $\mathbf{y}=\mathbf{z}$ or both $\langle \mathbf{y},\mathbf{x}\rangle \in \bigcup\{\mathbf{I_G},\mathbf{C_G},\mathbf{M_G},\mathbf{L_G}\}$ and $\langle \mathbf{z},\mathbf{x}\rangle \in \bigcup\{\mathbf{I_G},\mathbf{C_G},\mathbf{M_G},\mathbf{L_G}\}$; and

(3) For all $\mathbf{j} \in \mathbf{J_G}$, there are distinct $\mathbf{s}$, $\mathbf{s}'$, $\mathbf{s}''$ such that either $\mathbf{L_G s j}$, $\mathbf{L_G s}'\mathbf{j}$, and $\mathbf{R_G j s}''$ or $\mathbf{L_G s j}$, $\mathbf{R_G j s}'$, and $\mathbf{R_G j s}''$.

In Condition (3), in the first case **j** is known as a *join* and in the second case as a *branch*.

NOTE—Henceforth, "$\mathbf{E_G x y}$," "$\mathbf{I_G x y}$," etc., will usually be written instead of "$\langle \mathbf{x},\mathbf{y}\rangle \in \mathbf{E_G}$," "$\langle \mathbf{x},\mathbf{y}\rangle \in \mathbf{I_G}$," etc. Sometimes "$\mathbf{x} \rightarrow_{\mathbf{G}} \mathbf{y}$" will be used instead of "$\langle \mathbf{x},\mathbf{y}\rangle \in \mathbf{E_G}$." Where the graph **G** is understood, the subscript "$_{\mathbf{G}}$" on an arrow will sometimes be omitted. It will be assumed henceforth that the lexicon **L** is fixed.

As noted above, in terms of standard graphical IDEF0 diagrams, edges represent the connection itself, or "interface," between an incoming arrow segment and a box or junction, or a box or junction and an outgoing arrow segment. $\mathbf{I_G}$, $\mathbf{C_G}$, $\mathbf{O_G}$, and $\mathbf{M_G}$ represent of course the four different ways that an arrow segment can connect to a box. $\mathbf{L_G}$ represents the relation between an arrow segment **s** and a junction **j** when **s** connects to **j** "on the left," and $\mathbf{R_G}$ represents the relation between a junction **j** and an arrow segment **s** when **s** connects to **j** "on the right." The details of the definition place certain constraints on the structure of legitimate IDEF0 graphs. There are two sets of constraints.

The first set of constraints—the stipulation that $\mathbf{I_G}$, $\mathbf{C_G}$, $\mathbf{M_G} \subseteq \mathbf{S_G} \times \mathbf{B_G}$ and are pairwise disjoint, $\mathbf{O_G} \subseteq \mathbf{B_G} \times \mathbf{S_G}$, $\mathbf{L_G} \subseteq \mathbf{S_G} \times \mathbf{J_G}$, and $\mathbf{R_G} \subseteq \mathbf{J_G} \times \mathbf{S_G}$—constrains the pairs of vertices that can be connected by each type of edge: input, control, and mechanism edges must join arrow segments to boxes, output edges must join boxes to arrow segments, and junction edges must join arrow segments to arrow segments.

The second set of constraints—the listed Conditions (1) through (3)—are the minimal conditions on IDEF0 graphs that are imposed simply by the geometry of traditional IDEF0 diagrams. Thus, Condition (1) entails that an input arrow segment for a given box cannot also be an input (or control or mechanism) arrow segment for some other box; nor could it be an incoming segment to some junction. (Restricting $\mathbf{E_G}$'s functionality only to $\bigcup\{\mathbf{I_G},\mathbf{C_G},\mathbf{M_G},\mathbf{L_G}\}$, however, does allow boxes and junctions to have more than one outgoing arrow.) Condition (2) is the same condition in reverse for outgoing arrow segments, and thus entails that an output arrow segment for a given box cannot also be an output arrow for any other box. Condition (3) ensures that, for any given junction **j**, there must be either multiple arrow segments coming into **j** or multiple outgoing arrow segments going out of **j**, but not both. The first part of this condition ensures that junctions occur only when several arrow segments join or branch (so, in particular, there are no junctions involving only one segment on the left and one segment on the right), and the second part ensures that no junction can be both a branch and a join.

By the definition of $\mathbf{E_G}$, no arrow segment in an IDEF0 graph **G** is adjacent to any other. Since there are no edges in $\mathbf{E_G}$ whose first and second elements are both from $\mathbf{S_G}$, there are no elements of $\mathbf{S_G} \times \mathbf{S_G}$ in $\mathbf{E_G}$.

The following figures will be used to illustrate Definition 2 and the definitions introduced below:

— Figure 44 is a box-and-arrow skeleton of a simple IDEF0 diagram, with unique names assigned to boxes and arrow segments.

NOTE—These names are only tags for the distinct syntactic components of the diagram to illustrate the formal definitions being presented here; they are **not** to be thought of as names for concepts and activities associated semantically with these elements as in an actual IDEF0 diagram.)

— Figure 45 is a graph theoretic representation of that diagram. In the diagram, the type of an incoming or outgoing arrow segment is of course indicated geometrically by the point at which it joins the corresponding box. In the graph, junctions and arrow segments, like boxes, are represented as vertices, and the type of the arrow segment vertex with respect to an adjacent box or junction is indicated explicitly by a label on the edge connecting them.

Definition 3. Path

Let $\mathbf{G}=\langle \mathbf{V_G},\mathbf{E_G}\rangle$ be a graph. A *path* in **G** is a sequence of vertices (i.e., members of $\mathbf{V_G}$) $\langle \mathbf{y_1},...,\mathbf{y_n}\rangle$ such that for all $\boldsymbol{i<n}$ ($\boldsymbol{i>0}$), $\mathbf{y_i}\rightarrow\mathbf{y_{i+1}}$. The path is said to *traverse* each $\mathbf{y_i}$. The *length* of a path $\langle \mathbf{y_1},...,\mathbf{y_n}\rangle$ is $\boldsymbol{n-1}$.

Definition 4. Cycle

A path $\langle \mathbf{y_1},...,\mathbf{y_n}\rangle$ in **G** is a *cycle* iff $\boldsymbol{n>1}$ and $\mathbf{y_1}=\mathbf{y_n}$.

Definition 5. Successor

x is a successor (*predecessor)* of **y** *in* **G** iff there is a path from **y** to **x** (**x** to **y**). **x** is an *immediate* successor (predecessor) of **y** in **G** iff $\mathbf{x}\rightarrow\mathbf{y}$ ($\mathbf{y}\rightarrow\mathbf{x}$).

Definition 6. Complete path

A path $\mathbf{p}=\langle \mathbf{y_1},...,\mathbf{y_n}\rangle$ is *complete* in **G** iff $\mathbf{y_1}$ has no immediate predecessor in **G** and $\mathbf{y_n}$ has no immediate successor in **G**.

A path between two vertices ***N1*** and ***N2*** is just that: a way of getting from ***N1*** to ***N2*** by following edges, heeding their direction. Thus, in the graph in Figure 45, there is a path from ***s1*** to ***s4*** that, in addition to those two vertices, traverses ***b1***, ***s2***, ***b2***, ***s3***, and ***b3***. Formally, paths are represented simply by listing the sequence of vertices one must traverse along the path, beginning with ***N1*** and ending with ***N2***. A sequence $\langle N\rangle$ containing only a single vertex *N* is a perfectly good (if somewhat uninteresting) path of length 0; it will be important to bear this in mind for the definitions below. There can be more than one path from one vertex to another. In particular, if there is a cycle in path from ***N1*** to ***N2***, then there are infinitely many distinct paths from ***N1*** to ***N2***, because one can always cycle as many times as one pleases before finishing the path. The directed set $\langle \boldsymbol{b2},\boldsymbol{s3},\boldsymbol{b3},\boldsymbol{s5},\boldsymbol{j1},\boldsymbol{s7},\boldsymbol{j2},\boldsymbol{s9},\boldsymbol{b2}\rangle$ is a cycle in the graph in Figure 45. By definition, if **x** is any element in a cyclic path, then **x** is both a successor and a predecessor of itself.

Definition 7. Connected graph

Let $\mathbf{G}=\langle \mathbf{V_G},\mathbf{E_G}\rangle$ be a graph. Let $\mathbf{E^*}=\mathbf{E_G}\cup\mathbf{E_G^{-1}}$ (the set of arcs that connect the nodes in a graph), and let $\mathbf{G^*}=\langle \mathbf{V_G},\mathbf{E^*}\rangle$. **G** is *connected* iff, for all $\mathbf{x},\mathbf{y}\in\mathbf{V_G}$ (the nodes in the graph), there is a path in **G*** from **x** to **y**.

For example, there is no path from ***s1*** to ***s10*** in Figure 45. One can get from the former node ***s1*** to the latter node ***s10*** only by traveling “backwards” down one or more edges. Such a journey from one vertex to another—one that might involve traveling backwards down an edge—is called a *walk*. A walk is captured in the definition above essentially by adding, for each edge in the graph from ***N1*** to ***N2***, a corresponding edge from ***N2*** to ***N1***. This addition is achieved formally by “inverting” all the edges in $\mathbf{E_G}$ and results in the set $\mathbf{E_G^{-1}}$ and constructing a new graph **G*** with the same vertices. The edges of **G*** consist of the original edges **E** together with all the inverted edges $\mathbf{E_G^{-1}}$. (The *inverse* of $\langle \mathbf{x},\mathbf{y}\rangle$, of course, is $\langle \mathbf{y},\mathbf{x}\rangle$.) Intuitively, then, a graph **G** is connected if and only if there is a way to get from any vertex in **G** to any other by traveling either forward or backward along the edges in **G**.

Definition 8. Arrow path

An *arrow path* in **G** is a path $\langle \mathbf{y_1},...,\mathbf{y_n} \rangle$ such that $\mathbf{y_i} \in \mathbf{S_G} \cup \mathbf{J_G}$ ($\mathbf{1} \le \mathbf{i} \le \mathbf{n}$) and such that $\mathbf{y_1}, \mathbf{y_n} \in \mathbf{S_G}$. An *arrow* is an arrow path that it is not a subsequence of a larger arrow path.[7]

An arrow path in a graph is a path that traverses only arrow segments and junctions. Thus, $\langle s4 \rangle$, $\langle s6 \rangle$, $\langle s5, j1, s7 \rangle$, and $\langle s5, j1, s7, j2, s9 \rangle$ are all arrow paths. Only the first and last, however, are arrows, because, unlike the second and third arrow paths, neither is contained in a larger arrow path.

Definition 9. Boundary arrows and arrow segments

An outgoing (incoming) *boundary arrow* is an arrow whose last (first) arrow segment has no successor (predecessor) in **G**. A boundary arrow is either an outgoing or incoming boundary arrow. An outgoing (incoming) *boundary arrow segment* is an arrow segment that has no successor (predecessor) in **G**. A boundary arrow segment is an outgoing or incoming boundary arrow segment.

Thus, ***s4*** and ***s6*** are outgoing boundary arrow segments; ***s1***, ***s8***, and ***s10*** are incoming boundary arrow segments. $\langle s4 \rangle$ and $\langle s5, j1, s6 \rangle$ are outgoing boundary arrows, and $\langle s1 \rangle$ and $\langle s8, j2, s9 \rangle$ are incoming boundary arrows.

Definition 10. Incoming or outgoing

An arrow segment **s** in **G** is *incoming* (*outgoing*) with respect to a box or junction **e** in **G** iff **s** is an immediate predecessor (**s** is an immediate successor) of **e**. The arrow segment **s** is said to be *connected to* junction **e** iff it is either incoming or outgoing with respect to **e**. An arrow $\langle s1,\ldots,s_n \rangle$ in **G** is incoming (outgoing) with respect to a box **b** in **G** iff s_n is incoming (**s** is outgoing) with respect to **b**.

Thus, the boundary arrow $\langle s8, j2, s9 \rangle$ is incoming with respect to ***b2***, and the arrow $\langle s3 \rangle$ (hence also the arrow *segment* ***s3***) is outgoing with respect to ***b2*** and incoming with respect to ***b3***.

Definition 11. Node number

The numbers ***0*** to ***9*** and the infinite ordinal number **ω** are *node numbers*; and if ***n*** is a node number > 0 but $< \omega$, then $m(10^i)+n$ is a node number, where $1 \le m \le 9$ and ***i*** is the least number such that $10^i \le n$.

This definition of the notion of a node number is *recursive*; that is, ignoring **ω** for the moment, an initial base of instances of the notion is provided (the numbers ***0*** through ***9***) and then a recursive clause defines new instances of the notion in terms of instances already given. By iterating the recursive clause, then, one generates an infinite class of node numbers from the initial base class by, first, picking a node number $n > 0$ already in one's possession, finding the first power of ten greater than ***n***, multiplying that number by one of the initial node numbers (other than ***0***), and adding that product to ***n***. This yields the following sequence of numbers: 0, 1, 2, …, 9, 11, 12, …, 19, 21, 22, …, 29, …, 91, 92, …, 99, 111, 112, …, 119, …, 121, 122, …, 129, …, 191, 192, …, 199, 211, …, 299, …, 911, …, 999, 1111, 1112, … 1119, … .

[7] $\langle y_1,\ldots,y_n \rangle$ is a *subsequence* of $\langle x_1,\ldots,x_m \rangle$ if and only if for some i, j, $1 \le i,\ j \le m$, $\langle y_1,\ldots,y_n \rangle = \langle x_1,\ldots,x_{i-1} \rangle \hat{} \langle x_i,\ldots,x_j \rangle \hat{} \langle x_{j+1},\ldots,x_m \rangle$ (where $\langle z_1,\ldots,z_i \rangle \hat{} \langle w_1,\ldots,w_m \rangle$ is the concatenation of $\langle z_1,\ldots,z_i \rangle$ with $\langle w_1,\ldots,w_m \rangle$.

The convention in traditional IDEF0 diagrams is that a node number that uniquely designates a function within a given model is the result of affixing the letter "**A**" to the numerals corresponding to the above numbers. For purposes of abstract syntax it is cleaner just to use the numbers themselves and simply preserve the traditional practice as a usage convention. The standard IDEF0 node numbering scheme requires that the top level context diagram receive the special diagram number ***A-0***; but the success of this scheme relies on taking node numbers to be strings rather than genuine numbers (since ***-0=0***, and hence there is no distinction between ***A-0*** and ***A0*** unless one is talking about the strings "A-0" and "A0"). The point to note (for those familiar with IDEF0) is that, for the standard node numbering scheme to work—i.e., where the top level box is assigned box number ***0***, representing the function with the node number ***A0***, and its detail diagram has the diagram number ***A0***—the diagram number of the top level context diagram must be something other than ***0*** (and, of course, any other node number). Hence, it is convenient for purposes here simply to assign it the infinite number **ω**. The number ***–1*** would, in fact, have been the most natural choice, but the existing IDEF0 node numbering scheme permits the use of optional context diagrams, which are numbered with negative numbers. Such diagrams, and the requisite node numbering scheme, will be introduced in the next draft of this document.

B.2 IDEF0 diagram structure

In this clause the notion of a diagram structure is defined. It is important to note that it is the notion of a diagram *structure* that is defined, rather than the notion of a diagram per se. *Diagrams*, as the term is generally used, are components of actual IDEF0 models. They may contain names, notes, and other material that have no formal counterpart. A diagram structure, by contrast, captures only the *form* of the boxes and arrows in a diagram, a form that might be exhibited by many different actual IDEF0 diagrams. That pointed out, however, in the interest of brevity, the term *diagram* will be used to mean *diagram structure* hereafter.

B.2.1 Diagram prestructures

Definition 12. IDEF0 diagram prestructure

An *IDEF0 prestructure* $\mathbf{D}$ is a triple $\langle \mathbf{G}, \#_{\mathbf{n}}, \# \rangle$, where $\mathbf{G}=\langle \mathbf{V_G}, \mathbf{E_G} \rangle$ is an IDEF0 graph, and $\#$ is a function on $\mathbf{B_G}$ and $\#_{\mathbf{n}}$ is a function on $\mathbf{B_G} \cup \{\mathbf{G}\}$, such that

(1) For every $\mathbf{b} \in \mathbf{B_G}$, there is an $\mathbf{s} \in \mathbf{S_G}$ such that $\mathbf{C_G sb}$.

This requirement states that every box have at least one control arrow.

(2) For every $\mathbf{b} \in \mathbf{B_G}$, there is an $\mathbf{s} \in \mathbf{S_G}$ such that $\mathbf{O_G bs}$.

This requirement states that every box have at least one output arrow.

(3) Every box $\mathbf{b}$ in $\mathbf{B_G}$ is assigned a unique *box number* $\#(\mathbf{b})$ between ***0*** and ***9*** distinct from the box numbers of the other boxes in $\mathbf{B_G}$.

(4) Each box is assigned a node number $\#_{\mathbf{n}}(\mathbf{b})$.

(5) $\mathbf{G}$ is assigned a node number $\#_{\mathbf{n}}(\mathbf{G})$.

NOTE—Rules about labeling conventions for boxes and arrows have been omitted. Such conventions are important, but they are not appropriately treated as elements of an abstract syntax.

B.2.2 Diagram structures

Definition 13. Context diagram structure

If $\mathbf{D}=\langle \mathbf{G}, \#_{\mathbf{n}}, \# \rangle$ is a prestructure and in addition the following conditions hold, then **D** is a context diagram structure (or simply, *context diagram*):

(1) $\#_{\mathbf{n}}(\mathbf{G})=\omega$;

(2) **G** contains exactly one box **b**; and

(3) $\#_{\mathbf{n}}(\mathbf{b})=\#(\mathbf{b})=\boldsymbol{0}$.

(4) Every arrow segment in **G** is an arrow.

Traditional IDEF0 practice assigns the string "A-0" to the top-level context diagram of an IDEF0 model. As above, however, this label is unnecessary for the abstract syntax and can be viewed as a usage convention.

Definition 14. Noncontext diagram structure

If $\mathbf{D}=\langle \mathbf{G}, \#_{\mathbf{n}}, \# \rangle$ is a prestructure and the following conditions hold, then **D** is a *noncontext diagram structure* (*noncontext diagram*):

(1) $\boldsymbol{2} \leq \mathrm{card}(\mathbf{B}_{\mathbf{G}}) \leq \boldsymbol{9}$.

(2) For each box **b** in $\mathbf{B}_{\mathbf{G}}$, $\#(\mathbf{b}) \in \{\boldsymbol{1}, \ldots, \mathrm{card}(\mathbf{B}_{\mathbf{G}})\}$.

(3) For each box **b** in $\mathbf{B}_{\mathbf{G}}$, $\#_{\mathbf{n}}(\mathbf{b})=\boldsymbol{10}(\#_{\mathbf{n}}(\mathbf{G}))+\#(\mathbf{b})$.

Condition (1) is simply the 2-to-9 rule. Together with Condition (3) in Definition 12, Condition (2) requires that the boxes in a diagram structure $\mathbf{D}=\langle \mathbf{G}, \#_{\mathbf{n}}, \# \rangle$ be uniquely numbered from ***1*** to ***n***, where ***n*** is $\mathrm{card}(\mathbf{B}_{\mathbf{G}})$, i.e., the number of boxes in **D**. Condition (3) ensures that the node number of a box satisfies the IDEF0 node numbering conventions: $\#_{\mathbf{n}}(\mathbf{b})$ is the number denoted by the result of concatenating the numeral for the node number of **G** with the numeral for **b**'s box number. Because **b**'s box number is always between ***1*** and ***9***, inclusive, by Condition (5) in Definition 12, $\boldsymbol{10}(\#_{\mathbf{n}}(\mathbf{G}))+\#(\mathbf{b})$ will always be a legitimate node number [as $\boldsymbol{10}(\#_{\mathbf{n}}(\mathbf{G}))$ simply "shifts" the places in the decimal numeral for $\#_{\mathbf{n}}(\mathbf{G})$ one place "to the left"], and, therefore, this condition is well-founded.

Definition 15. IDEF0 diagram structure

An *IDEF0 diagram structure* (or simply, *diagram*) is either a context diagram structure or a noncontext diagram structure.

B.3 IDEF0 model structures

In this clause the notion of an IDEF0 model structure is defined. The term "model structure" is chosen for reasons analogous to the choice of "diagram structure" in the previous clause: a model *structure* characterizes a certain form that can be shared by many actual IDEF0 models.

NOTE—Henceforth, the IDEF0 graph **G** in a diagram **D** will not be distinguished from the diagram itself. Thus, for example, for a diagram **D**, "$\#(\mathbf{D})$" will be written instead of "$\#(\mathbf{G})$" to indicate the node number of **D**'s graph, "$\mathbf{B}_{\mathbf{D}}$" instead of "$\mathbf{B}_{\mathbf{G}}$" to indicate the boxes in the graph, and so on.

Definition 16. Parent and child diagram relationships

For any two diagrams **D** and **D´**, **D** is a *parent diagram of* (or simply, *parent of)* **D´** iff there is a box **b** in **D** such that $\#_n(\mathbf{b})=\#(\mathbf{D´})$. In such a case, **D´** is said to be a *child* of **D**, **D´** a *detail diagram of* **b**, and **b** a *parent box* of **D´**.

The parent relation is thus defined simply in terms of node numbers. Specifically, a diagram **D** is a parent of some other diagram **D´** if and only if some box in **D** has the same node number as **D´**.

Before giving the next definition, it will be convenient to extend some of the notational conventions we have adopted for graphs to diagrams. If $\mathbf{D}=\langle \mathbf{G},\#_n,\# \rangle$ is a diagram, such that $\mathbf{G}=\langle \mathbf{V_G},\mathbf{E_G} \rangle$, $\mathbf{B_G}=\mathbf{V_G}\cap\mathbf{B}$, $\mathbf{S_G}=\mathbf{V_G}\cap\mathbf{S}$, and $\mathbf{J_G}=\mathbf{V_G}\cap\mathbf{J}$, then let $\mathbf{B_D}=\mathbf{B_G}$, $\mathbf{S_D}=\mathbf{S_G}$, and $\mathbf{V_D}=\mathbf{V_G}$.

Definition 17. Diagram uniqueness

Two diagrams **D**, **D´** are *disjoint* iff $\mathbf{B_D}\cap\mathbf{B_{D´}}=\mathbf{S_D}\cap\mathbf{S_{D´}}=\mathbf{J_D}\cap\mathbf{J_{D´}}=\varnothing$.

D and **D´** are disjoint, that is, if and only if they have no elements of **B** *(boxes)* or **S** *(arrow segments)* or **J** *(junctions)* in common. This definition is needed because, since **B** and **S** and **J** are simply sets of primitive elements for building diagrams, there is no reason in general why the same syntactic element should not be used in more than one diagram.

Definition 18. Ancestral tree

An *ancestral tree* is a tree $\Delta=\langle \delta,\mathbf{P} \rangle$ such that δ is a set of pairwise disjoint diagrams, **P** is the parent-of relation, and for all $\mathbf{D},\mathbf{D´}\in\delta$, $\#_n(\mathbf{D})\neq\#_n(\mathbf{D´})$.

A tree is a directed graph **T** because there is a single vertex—the root—with no predecessors and there is a unique path from the root to every other vertex of **T**. (Therefore, all trees are acyclic, i.e., contain no cycles.) Vertices with no successors are called *leaves*. These directed graphs are called trees because, letting dots represent vertices and arrows represent edges in the usual fashion, they typically have a distinctively arboreal appearance, though usually inverted, as shown in Figure 46.

$\Delta=\langle \delta,\mathbf{P} \rangle$ is an *ancestral* tree, then, if and only if its vertices are diagrams, each vertex is a parent of its immediate successors in Δ (this relationship is guaranteed by letting the edges of the tree be the parent-of relation), and no two diagrams share the same node number. (This last condition prevents the possibility of more than one detail diagram for a given box.)

IDEF0 models will be defined formally as ancestral trees that satisfy certain conditions. First some auxiliary notions need to be established.

Definition 19. Ancestral box

Let Δ be an ancestral tree and let **b** be a box in Δ and **D** a diagram in Δ. Then **b** is said to be an *ancestral box of* **D** *in* Δ iff **D** is a detail diagram of **b** or **b** has a detail diagram that contains an ancestral box of **D** in Δ.

That is, **b** is an ancestral box of **D** in an ancestral tree if and only if **D** is a detail diagram of **b**, or a detail diagram of a box in a detail diagram of **b**, or a detail diagram of a box in a detail diagram of a box in a detail diagram of **b**, etc.

Definition 20. ICOM coding

Let Δ be an ancestral tree and let $\mathbf{D_{root}}$ be the root of Δ. Let β be the set of boundary arrow segments in $\Delta-\mathbf{D_{root}}$ (all the diagrams in a model except the model's ***A-0*** context

diagram) and let γ be the set of arrow segments in Δ. An ICOM coding for Δ is a total function $\boldsymbol{c}$ on β into γ such that for any **s** in β, if **s** is incoming (outgoing), then $\boldsymbol{c}$(**s**) must be incoming (outgoing) with respect to some ancestral box of **D** in Δ. The pair $\langle\Delta,\boldsymbol{c}\rangle$ is called a *coded* ancestral tree.

The purpose of an ICOM coding is to relate boundary arrows on a child diagram to arrows connected to its parent box. The traditional approach to ICOM coding in IDEF0 depends on the spatial geometry of the arrows in a parent diagram. This method, of course, depends on the particular features of the standard graphical representation and, therefore, does not apply to the abstract syntax. To capture the function of ICOM coding abstractly, the notion is defined simply as a function of the set of boundary arrow segments in all the diagrams of an ancestral tree Δ—except for the root diagram $\mathbf{D}_{\mathbf{root}}$—subject to three conditions. First (Condition (i)), whenever the mapping is defined on some incoming or outgoing boundary arrow segment **s** in a diagram **D** of Δ, the mapping must take **s** to a segment that is incoming or outgoing, respectively, with respect to an ancestral box of **D**. In the typical, nontunneled case (see Definition 21), the ancestral box in question will be the parent box. This definition permits shifting of input, control, and mechanism arrow roles from parent diagram to child diagram.

The second condition is best explained after another definition. As noted, the usual notion of an ICOM coding only relates boundary arrows on a child diagram to arrows connected to its parent box. However, the definition departs from the traditional approach slightly by generalizing the usual notion in a manner that also yields a definition of *tunneling*.

Definition 21. Tunneling

Let $\Delta_{\boldsymbol{c}}=\langle\Delta,\boldsymbol{c}\rangle$ be a coded ancestral tree. Let **b** be a box in $\Delta_{\boldsymbol{c}}$ and suppose that **b** has a detail diagram **D** in $\Delta_{\boldsymbol{c}}$. A boundary segment **s** in **D** is *tunneled upward* iff $\boldsymbol{c}$(**s**) is not connected to **b**. Conversely, an arrow segment **s** connected to **b** is *tunneled downward* iff there is no boundary segment **s**′ in **D** such that $\boldsymbol{c}$(**s**′)=**s**, *or* there is a diagram **D**′ such that **b** is not the parent of **D**′, and for some arrow segment **s**′ in **D**′, $\boldsymbol{c}$(**s**′)=**s**.

A boundary arrow segment **s** in a detail diagram **D** for a box **b** is tunneled upward (i.e., “at the unconnected end”), if and only if the ICOM coding $\boldsymbol{c}$ for the ancestral tree Δ does not correlate it with any arrow segment connected to **b**, but rather to a segment connected to some other ancestral box. (That $\boldsymbol{c}$ must map **s** to an arrow segment connected to some other ancestral box follows from the definition of a coded ancestral tree.) Conversely, an arrow segment **s** that is connected to **b** is tunneled downward (i.e., “at the connected end”) if and only if $\boldsymbol{c}$ does not correlate any boundary arrow in **D** with **s**, but rather only (at most) correlates a boundary arrow in some lower level diagram **D**′ with **s**. Again by the definition of a coded ancestral tree, **b** must be an ancestral box of **D**′. Within the IDEF0 syntax proper, the generalized definition of an ICOM fulfills the task—traditionally assigned to model notes—of identifying the “other end” of an upwardly tunneled arrow.

The qualification “at most” in the previous paragraph reflects the fact that an ICOM coding as defined above need not be *onto*. That is, where **b** is a box with a detail diagram, not every arrow segment adjacent to **b** need be correlated with a boundary arrow segment in some lower level diagram by an ICOM coding $\boldsymbol{c}$. More colloquially, an arrow segment can be tunneled downward without any reference to the “other end” of the tunnel. By contrast, all arrows that are tunneled upward are required to be correlated with an arrow segment that is adjacent to some ancestral box. (This requirement is enforced in the syntax by requiring an ICOM coding to be a *total* function on the class of boundary arrows in a model excluding those in the context diagram of the model.) For an input, control, mechanism, or output that has been identified in some lower level activity to have no trace at a higher level would be a modeling error.

The definition also allows *multiple* references to a given arrow segment by arrow segments in lower level diagrams because arrows may branch while tunneled. For example, input data for an activity at a higher, coarser level of representation might on closer analysis be seen to "feed" numerous lower level activities.

The account of ICOM coding developed here departs slightly in two ways from the traditional approach, which defines tunneling for boundary arrows generally. First, tunneling has been defined for arrow *segments* instead of arrows. A detail diagram for box ***b1*** in Figure 47, for instance, can be sensitive at most to the outputs indicated by ***s1*** and ***s2***; the branching of ***s2*** into ***s3*** and ***s4*** indicates a division of data/objects that occurs outside the activity indicated by the box, and hence outside the scope of a detail diagram for the box.

An IDEF0 *model structure* can now be defined simply to be a special sort of coded ancestral tree:

Definition 22. IDEF0 model structure

Let Δ_c be a coded ancestral tree. Then Δ_c is an *IDEF0 model structure* in the lexicon $\mathbf{L}$ iff

(1) $\mathbf{D}_{root}$ is a context diagram;

(2) For all $\mathbf{D} \in \Delta - \mathbf{D}_{root}$, $\mathbf{D}$ is a noncontext diagram.

To be a full-blown IDEF0 model structure, a coded ancestral tree must satisfy two simple conditions: the root of the tree must be a context diagram, and every other diagram in the tree must be a noncontext diagram. Intuitively, every noncontext diagram represents the decomposition of the function indicated by some parent box. Hence, a noncontext diagram within an IDEF0 model structure is usually referred to as a *decomposition* diagram.

The definition is illustrated in Figure 48, which shows the following graphical IDEF0 model structure (albeit with only two boxes in the noncontext diagrams for the sake of simplicity):

Obviously the pair $\Delta = \langle \delta, \mathbf{P} \rangle$ is a directed graph, where δ is the set consisting of the IDEF0 graphs $\{\mathbf{D0}, \mathbf{D1}, \mathbf{D2}\}$, and $\mathbf{P}$ is the set of edges $\{\langle \mathbf{D0},\mathbf{D1} \rangle, \langle \mathbf{D1},\mathbf{D2} \rangle\}$ (indicated by the dashed lines). Δ is also clearly a tree, where **D0** is the root diagram. Given the node numbering indicated in the lower left corners of boxes and diagrams, $\mathbf{P}$ satisfies the definition of the parent-of relation (Definition 16), and hence, by Definition 18 (since in addition no two diagrams have the same node number and the diagrams in δ are pairwise disjoint (Definition 17)), Δ is an ancestral tree. The mapping ***c*** indicated to the right of **D1** and **D2** satisfies the conditions on being an ICOM coding set down in Definition 20 (e.g., the outgoing boundary arrow ***s13*** is mapped to an arrow segment that is outgoing with respect to **D2**'s parent box, which is, therefore, an ancestral box of **D2**[8]), and hence the pair $\langle \Delta, c \rangle$ (also known as Δ_c) is a coded ancestral tree, by Definition 21. Since the root **D0** of Δ satisfies the definition of a context diagram, and since **D1** and **D2** do not, it follows by Definition 22 that Δ_c is indeed an IDEF0 model.

B.4 Scope of this annex

Readers familiar with IDEF0 will notice the absence of several common notions, in particular, model identifiers, higher level context diagrams, and call arrows. Though all three are important, they have not been included in the formal syntax because they are *theoretically* dispensable. A call arrow, for instance,

[8]Note also that ***s11*** is tunneled upward and, hence, that its "other end", ***s2***, is tunneled downward.

could *in principle* be replaced by the box that it points to, together with its "descendants" (suitably modified, perhaps). For similar reasons, labeling conventions for arrow segments have also not been included. Every arrow segment is simply assumed to have a label in this annex.

The absence of higher level optional context diagrams should also be noted. It is not clear that such diagrams should be thought of as theoretically dispensable. However, their inclusion in this standard would add a significant degree of complexity that it was felt would best be avoided in the first version of this standard. Future versions will be suitably modified to indicate how such diagrams can be included in legitimate IDEF0 model structures.

Annex C

(informative)

Examples of IDEF0 usage and style

This annex contains the figures cited in the body and in Annex B of this standard.

Each figure referenced in the text is presented as an FEO page in this annex. Most such FEO pages are accompanied by one or more text pages and some are accompanied as well by one or more glossary pages.

C.1 Overview

These figures for the language formalization are designed to demonstrate technical aspects of IDEF0 models and model pages as standardized by IEEE Std 1320.1-1998 and to illustrate concepts developed by Annex B. In addition, these figures are also designed to demonstrate preferred IDEF0 style and presentation. To support these objectives, these figures use the IDEF SDF.

Because the SDF, with identifying, message, and working information fields, is formatted for ordinary 8½-by-11-inch paper, the figures have been collected into their own annex of this document. This separate compilation avoids these problems:

a) *Poor readability.* The SDF claims an entire page each time it is used. Particularly when several figures are referenced from locations close to one another in the text, the flow of reading could be interrupred by several pages of illustrations.

b) *Absence of facing text and glossary pages.* A demonstration of good style and presentation within a document treating IDEF0 should include the use of facing text pages and glossary pages. Each figure is designed to illustrate a specific topic addressed in the body of the standard. However, these figures often contain features, unrelated to that primary purpose, that should be discussed in a nonbinding way. To include such nonbinding commentary in the body of a binding standard would be inappropriate. In addition the use of facing text and and glossary pages to accompany and illuminate each formal figure clearly could only compound the first two problems.

c) *Implication of SDF standardization.* The SDF is not a subject of IEEE Std 1320.1-1998 nor has the SDF been standardized by any other standards document. Nonetheless, competent IDEF modelers routinely and customarily used the SDF in a standard way. Therefore, collecting the figures separately from the body of IEEE Std 1320.1-1998—which expresses the mandatory features and characteristics of IDEF0 models and model pages—distinguishes the nonbinding use of the SDF for IDEF modelers while allowing the presentation of conventional characteristics of IDEF0 model presentation.

C.2 About the IDEF SDF

The SDF is documented in the informative annexes of FIPS PUB 183 [B2] and FIPS PUB 184 [B3], among other sources. While the design and use of the SDF is well-established, neither design nor use has been formally standardized. It has been adapted here to the specific purpose of illustrating the language formalization. Indeed, the IDEF community remains reluctant to standardize the SDF precisely to avoid compromising the modeler’s capability to adapt the SDF to special purposes such as this document.

C.3 How the SDF is used in this document

To adapt the SDF to this document, the conventional fields of the SDF have been treated as described in this clause. Each field is discussed as it is found in the SDF, working from left to right and from top to bottom, through the working information and the identification fields.

C.3.1 Used at field

The *used at* field is not used.

C.3.2 Author field

The *author* field does not identify the individual who prepared these diagrams. Instead, the *author* field identifies the "IEEE IDEF0 Working Group," which directed the creation of these figures.

C.3.3 Project field

The *project* field identifies "IDEF0 Language Formalization" as the project name.

C.3.4 Model field

The language formalization introduces the requirement that a model's name and model name abbreviation be presented in the ***A-0*** context diagram of a model. Reinforcing this requirement, a field labelled *Model* has been added to the working information fields to capture the model name and its abbreviation. The model name abbreviation is given within parentheses following the model name. For the figures in this annex, the model name is given as "Formalization Figures" and "FF" is given as the model name abbreviation.

C.3.5 Notes field

Because these figures are not presented as examples of products produced during the development or kit cycle review of a model, the *notes* field is not used.

C.3.6 Status field

The status for each figure has been set to "Publication," which is consistent with the use of the page subfield within the number field (see *Number Field*, below). However, in spite of this publication status, all figures retain their working information fields. Thus, no cutting gutter between the working information fields and the message field is provided.

C.3.7 Reader/date field

Because these figures are not presented as examples of products produced during the development or kit cycle review of a model, the *reader/date* field is not used.

C.3.8 Context field

The language formalization does not address the notion of context represented by the *context* field of the SDF. In standard practice, the *context* field provides some sort of thumbnail image of a diagram's parent diagram or contains the appropriate bounding token for a model's highest context diagram(s).

Within an IDEF0 model, a nondiagram page has the same context as the IDEF0 diagram to which it is related. The difficulty here is that these figures are not necessarily related to specific IDEF0 diagrams because these figures do not constitute an IDEF0 model. Thus, a proper source for the contents of the *context* field is missing. In the absence of diagram contexts, the *context* fields of these figures are generally left blank. However, when a figure does present a diagram page or a fragment of such a diagram page, the *context* field will contain an appropriate context image or token.

C.3.9 Model page field

The traditional *node* field has been renamed. The new name, *model page*, reflects the language formalization's refinement of the FIPS PUB 183 concept of *node reference* and the introduction of new *page reference* terminology (see C.4).

C.3.10 Figure field

The traditional *title* field has been renamed the *figure* field and is used to present a figure caption rather than a proper IDEF0 diagram title. As established by the language formalization, the title of an IDEF0 diagram is (generally) the name of the box that the diagram details, and this diagram title is normally recorded in the SDF's *title* field. According to the language formalization, an IDEF0 diagram page is identified by both a title and a diagram number while an associated nondiagram model page is separately identified only by the page reference that links that page to its associated diagram.

The difficulty here is that these figures are not necessarily related to specific IDEF0 diagrams because these figures do not constitute an IDEF0 model. Thus, a proper source for the contents of the *title* field is missing. In the absence of diagram titles, the *figure* field is used to caption the contents of the SDF's message field, using MS Word's captioning construct.

C.3.11 Number field

The *number* field in these figures contains a C-number (the so-called *configuration control number*) and a *page* subfield with the label *P*. This *page* subfield contains a document page number determined by MS Word pagination.

C.4 Page references

While these figures illustrate features of IDEF0 models and model pages, these figures are not themselves an IDEF0 model; this annex contains only FEO, text, and glossary model pages. However, the language formalization specifies that every nondiagram page within an IDEF0 model must be associated through its page reference to one specific IDEF0 diagram.

In the absence of IDEF0 diagrams, semantically valid page references—in the sense of the language formalization—cannot be constructed for these FEO, text, and glossary pages. Nonetheless, even in the absence of a proper IDEF0 model, these figures should still demonstrate proper form, i.e., a syntactically valid page reference, in the model page field. To this end, fictitious diagram numbers have been derived from a topical sequencing of figures used in earlier versions of this document. Should a series of FEO

pages address a single topic, their page references will contain the same diagram numbers while their page type sequence numbers will increment.

To emphasize that this diagram reference is a syntactic construct rather than a semantically meaningful reference, “S” (for “subject”) is used as the node letter instead of the customary node letter “A,” as allowed by the language formalization. Thus, the first FEO page is identified as Figure 1 in the body of the standard and its model page reference is given as ***S1F1***. The second FEO page addressing the 19th subject addressed by FEO pages in the document would be similarly given the page reference ***S19F2***. Likewise, a facing text page that discusses the first FEO page discussing the fifth topic for which FEO pages have been provided would be given the page reference ***S5F1T1***.

The language formalization allows as many as nine boxes in an IDEF0 diagram; this new flexibility allows the digit “9” in the FEO page reference ***S19F2*** to be syntactically correct. However, the digit “0” would remain syntactically incorrect in these constructed diagram references; this is because the box number ***0*** is allowed only to identify the box that represents the ***A0*** function in the ***A-0*** and ***A-1*** context diagrams. Thus a constructed page reference such as ***S20F1*** would be both semantically invalid and syntactically incorrect. Therefore, the topical page references constructed for the FEO pages of this annex do not include fictitious diagram numbers which include the digit ***0***; in effect, the sequence of subject numbers standing in for diagram numbers skips “10,” “20,” and “30.”

C.5 Facing text pages

The content of facing text pages in this annex describes and explains the FEO figures that they accompany. The content of these text pages is informative only.

C.6 Glossary pages

The content of glossary pages in this annex define terms in the FEO figures that they accompany. The content of these glossary pages is informative only, unless the definition is also given in Clause 2.

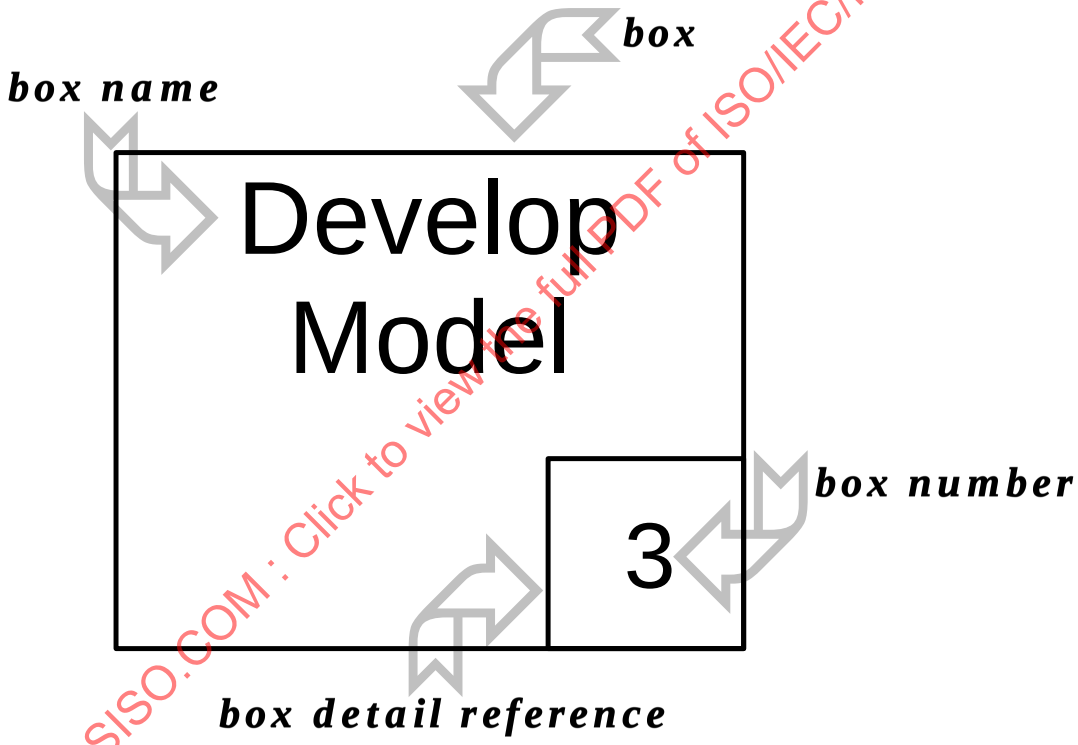
C.7 Diagram.feature references

To minimize visual clutter, IDEF0 *diagram.feature* references are used extensively in these figures rather than activity names and arrow labels. Refer to facing text pages for more expansive information.

C.8 Acknowledgments

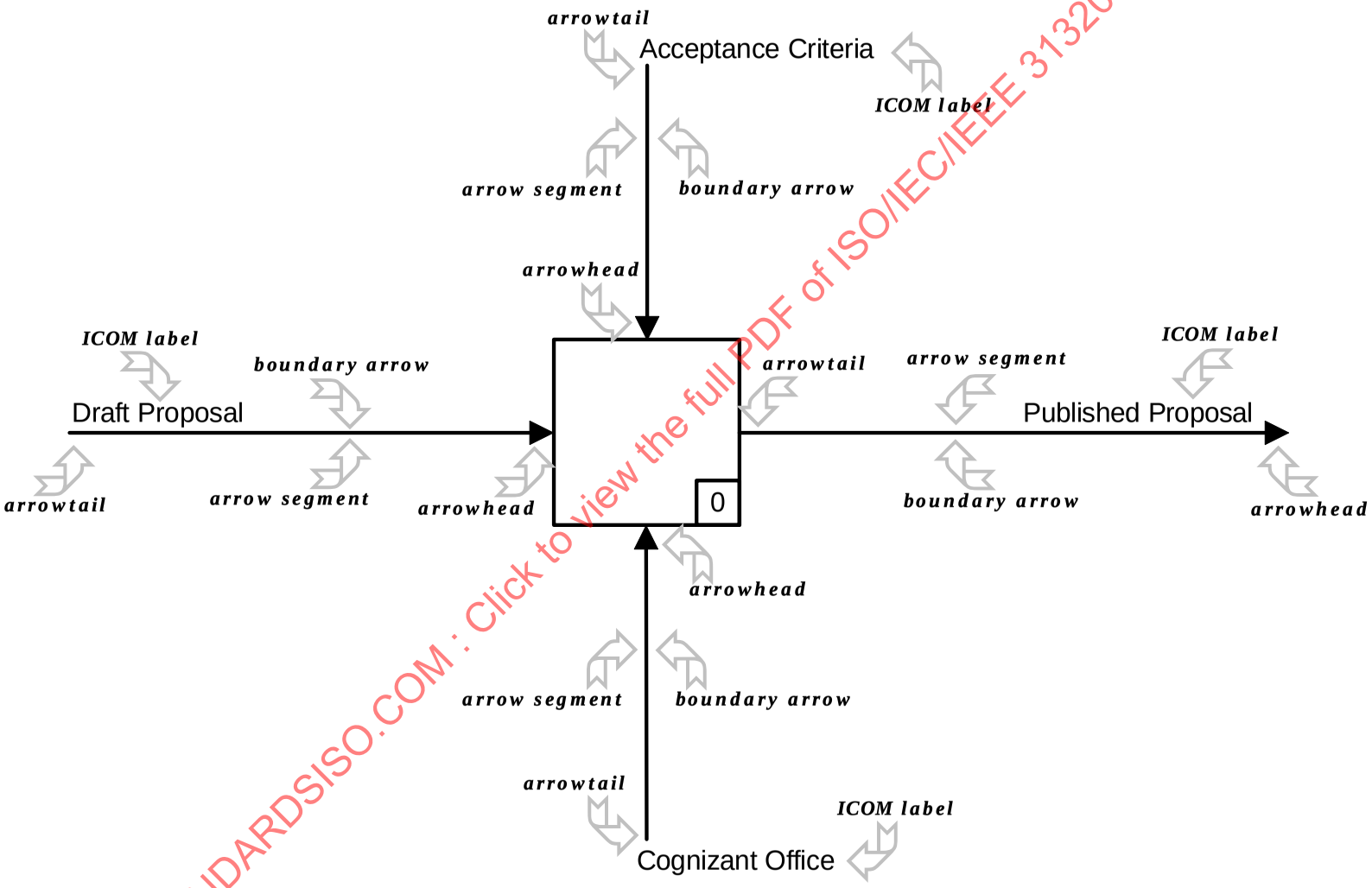
These figures were prepared using Design/IDEF, Release 3.7, from Meta Software, Inc.

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



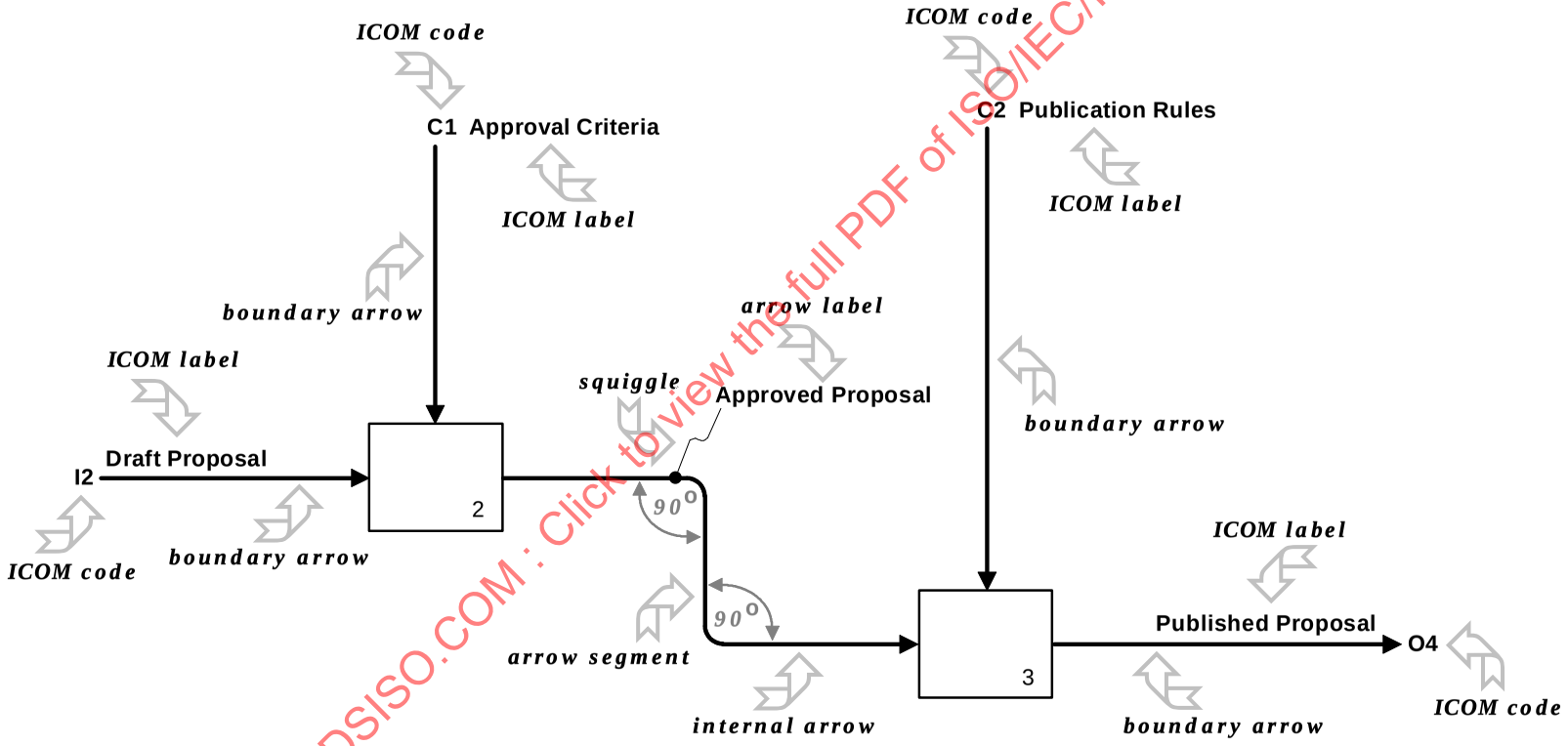
Model Page: FF/S1F1	Figure 1—Box Syntax	Number: AKB001	P. 58

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



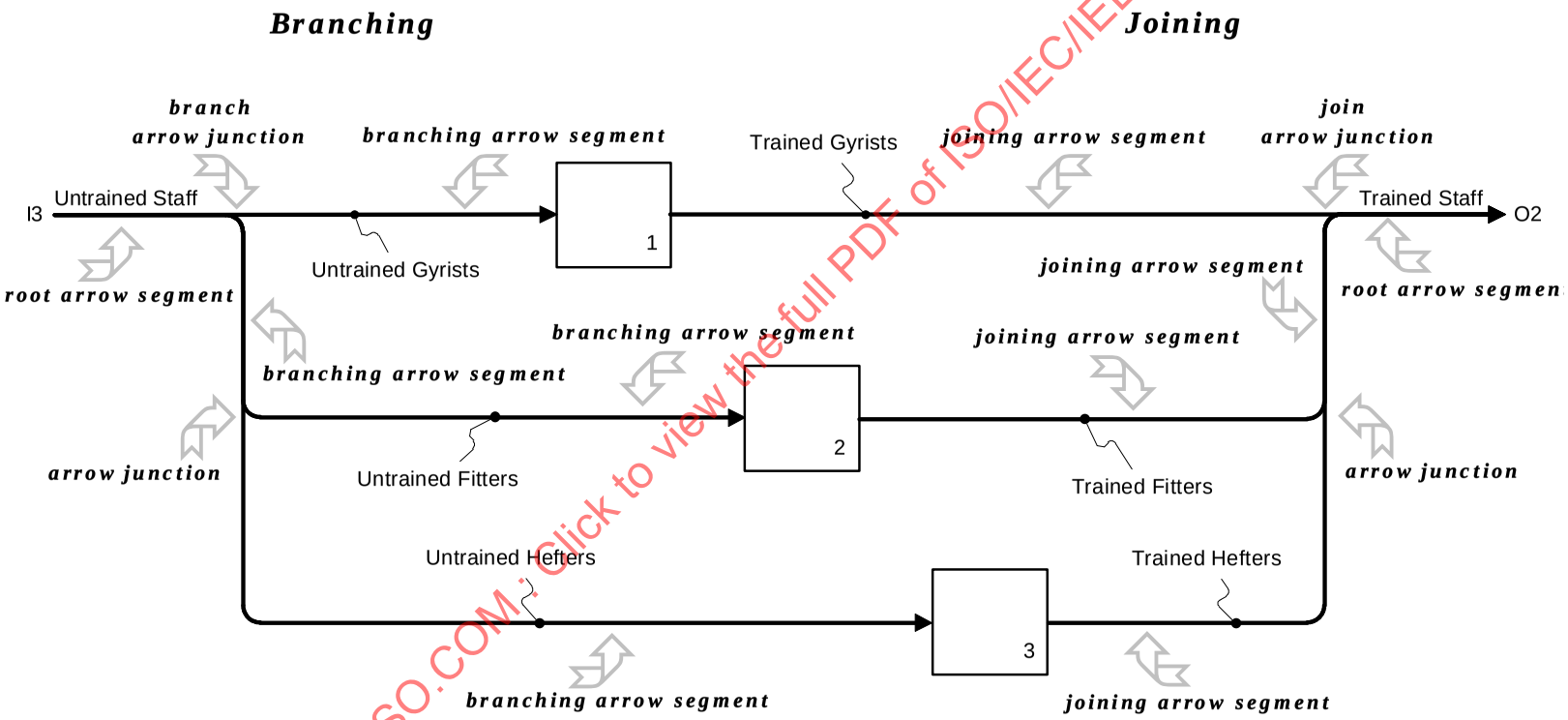
Model Page: FF/S2F1	Figure 2—Arrow Syntax (A-0 Context Diagram)	Number: AKB002	P. 59

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



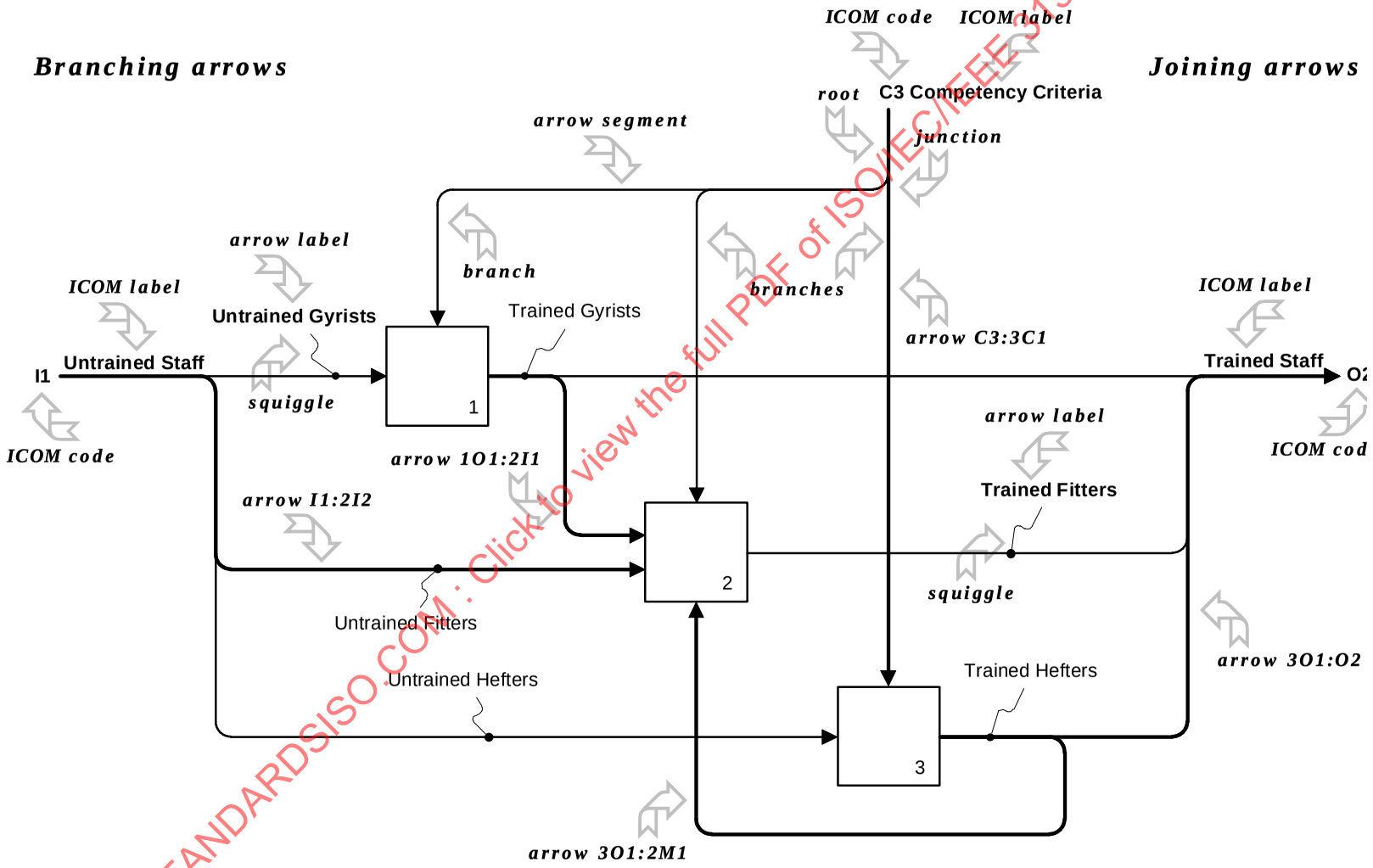
Model Page: FF/S2F2	Figure 3—Arrow Syntax (Decomposition Diagram)	Number: AKB003	P. 60

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



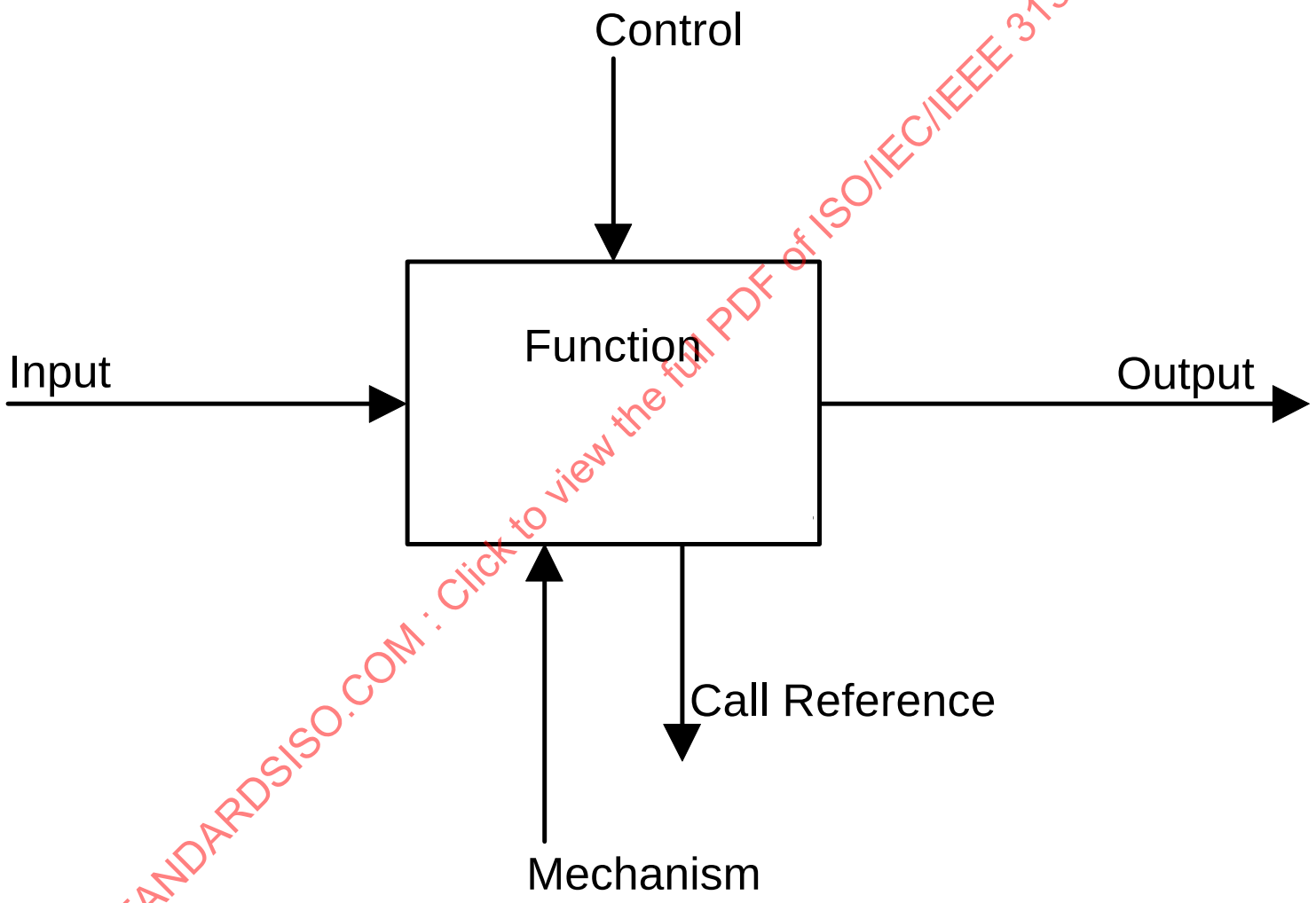
Model Page: FF/S2F3	Figure 4—Arrow Syntax (Arrow Segments)	Number: AKB004	P. 61

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



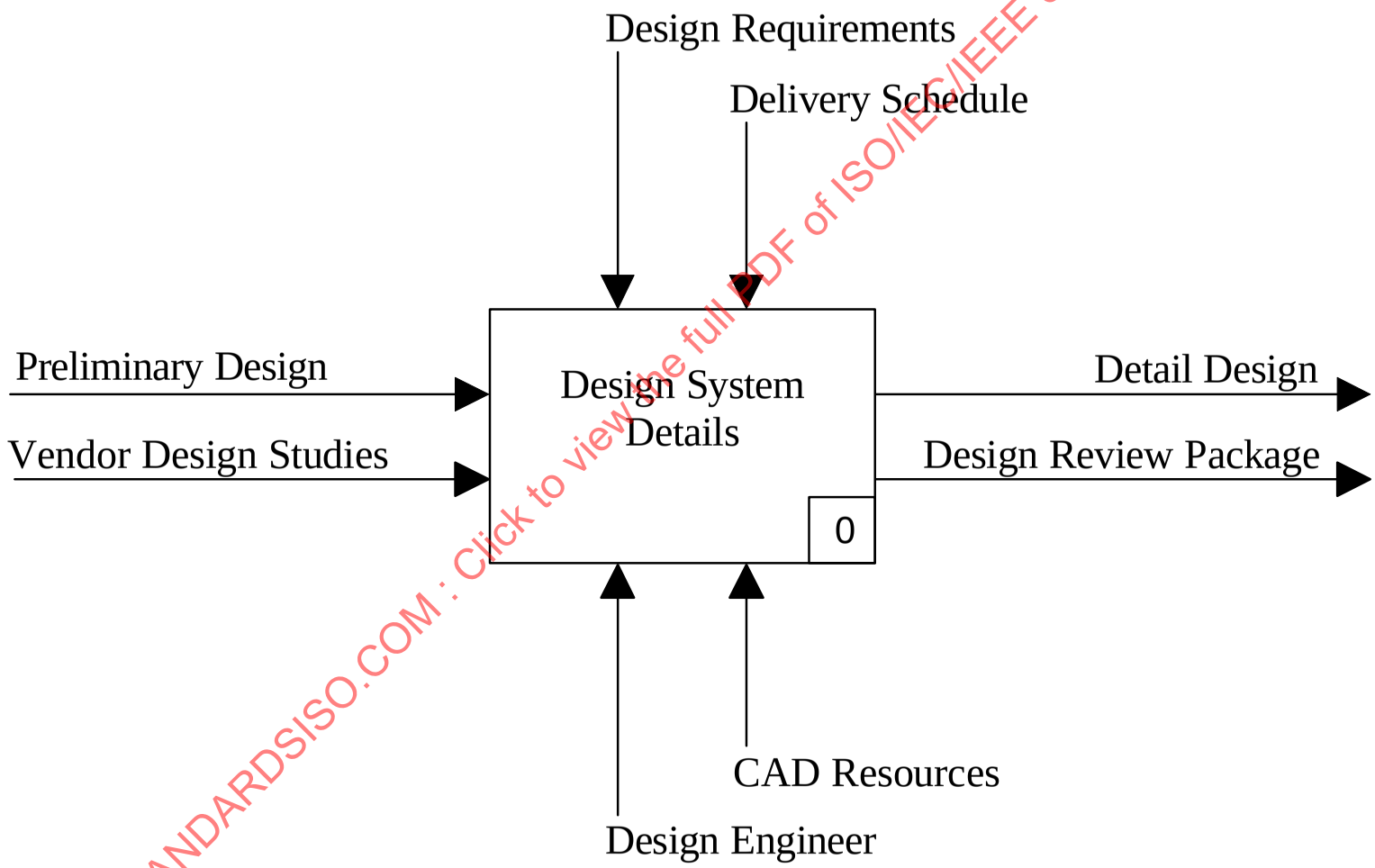
Model Page: FF/S2F4	Figure 5—Arrow Syntax (Multi-Segment Arrows)	Number: AKB005	P. 62

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



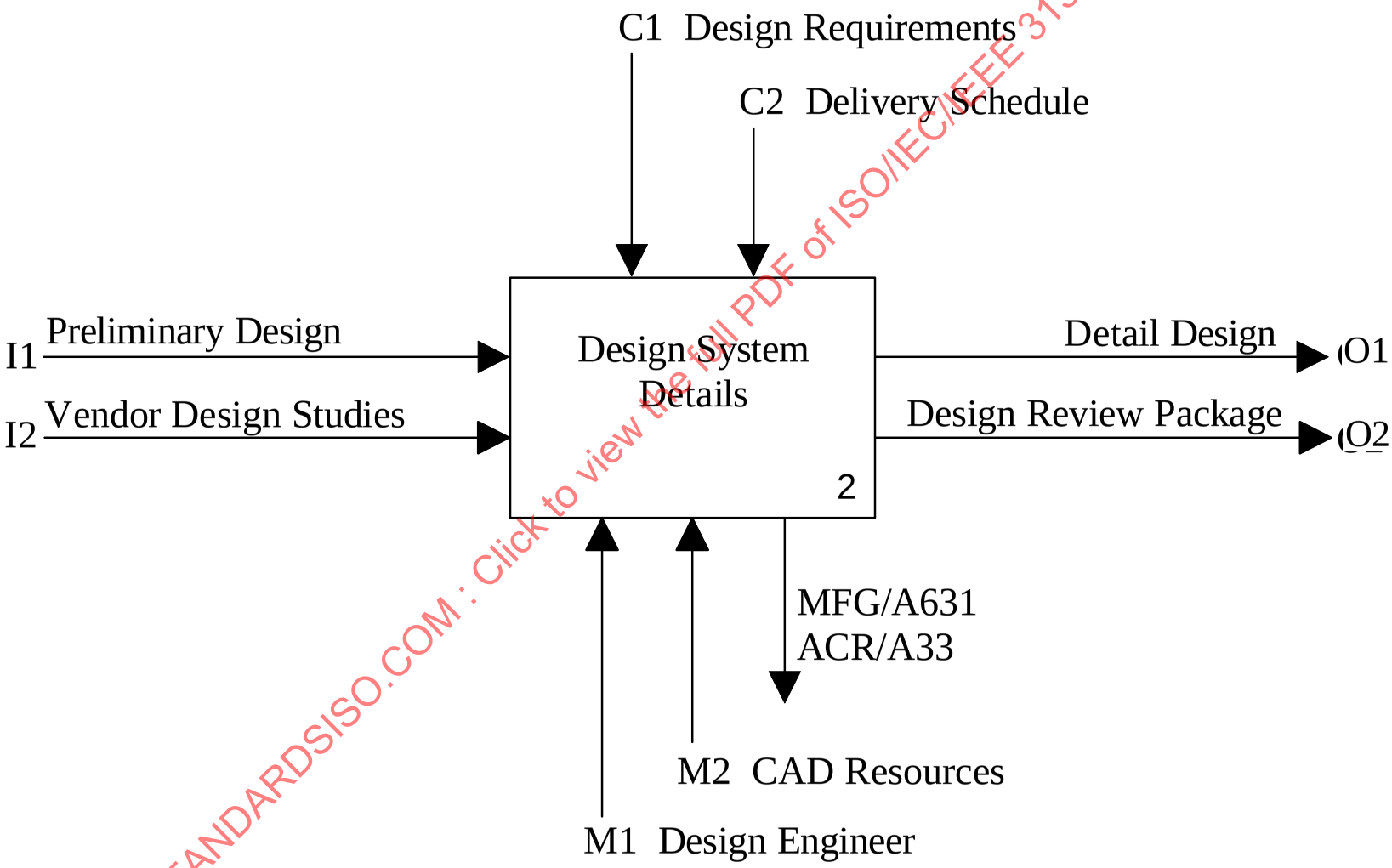
Model Page: FF/S3F1	Figure 6—Arrow Attachments and Roles	Number: AKB006	P. 63

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



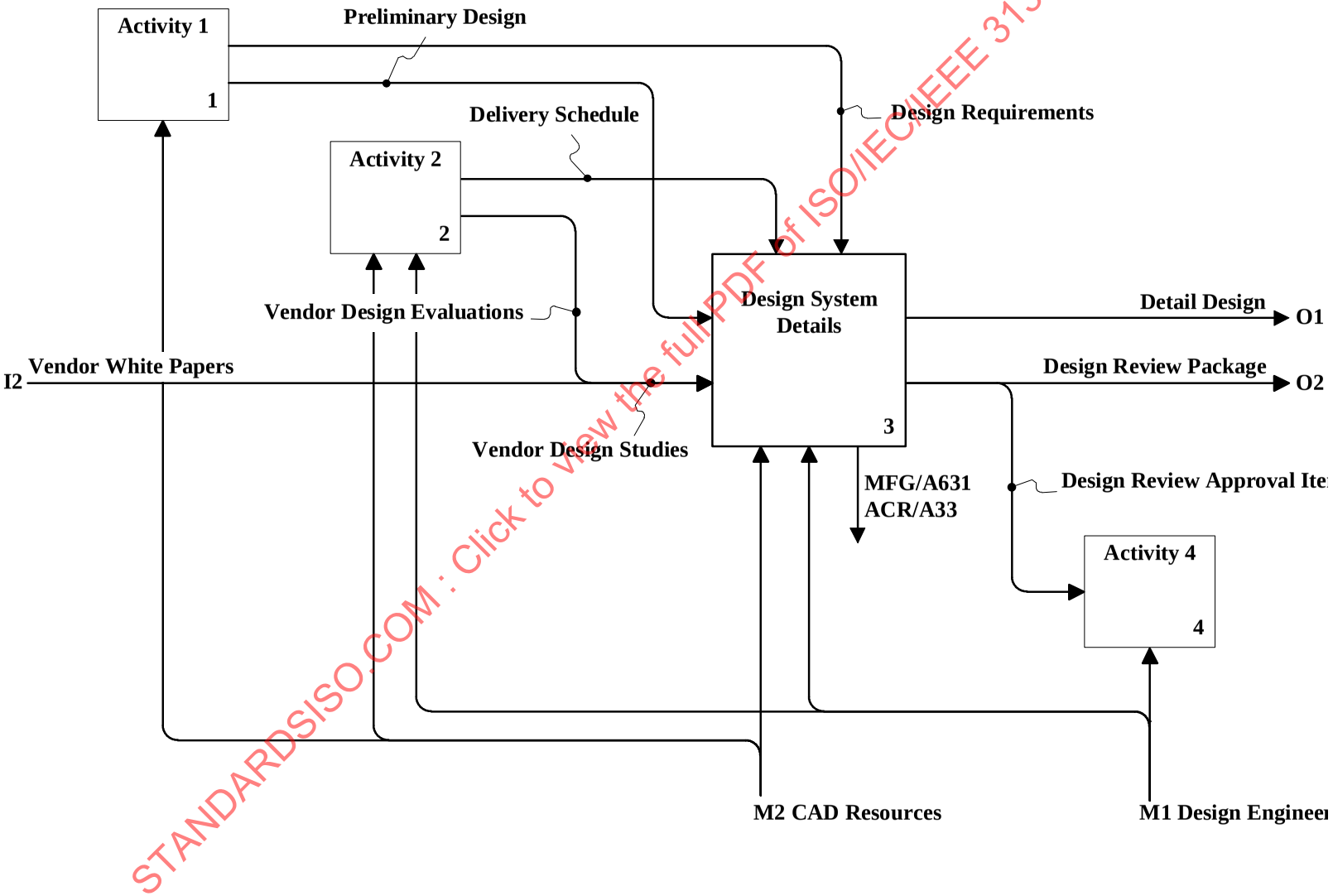
Model Page: FF/S3F2	Figure 7—Label and Name Example (A-0 Context Diagram)	Number: AKB007	P. 64

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



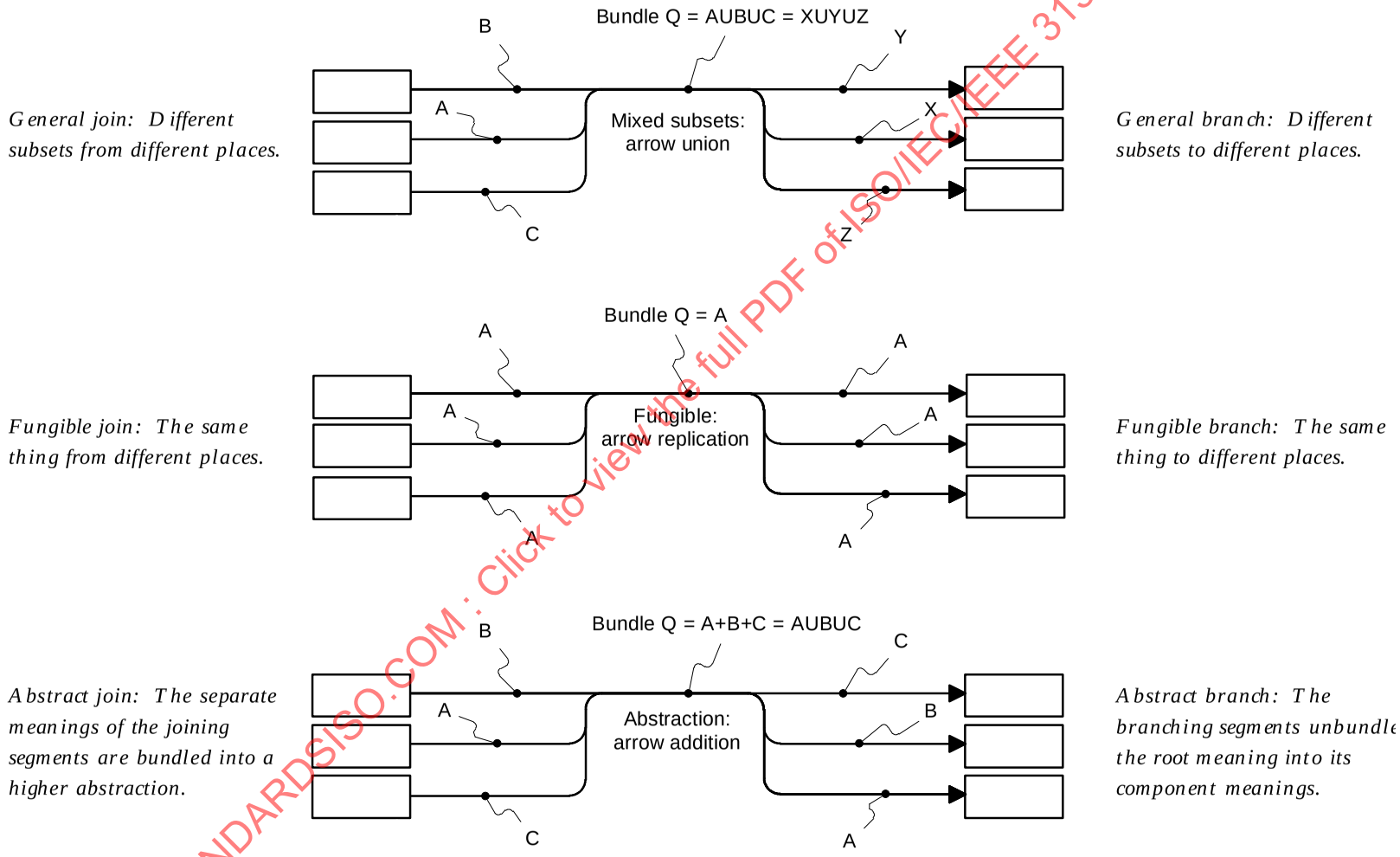
Model Page: FF/S3F3	Figure 8—Label and Name Example (Boundary Arrows)	Number: AKB008	P. 65

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



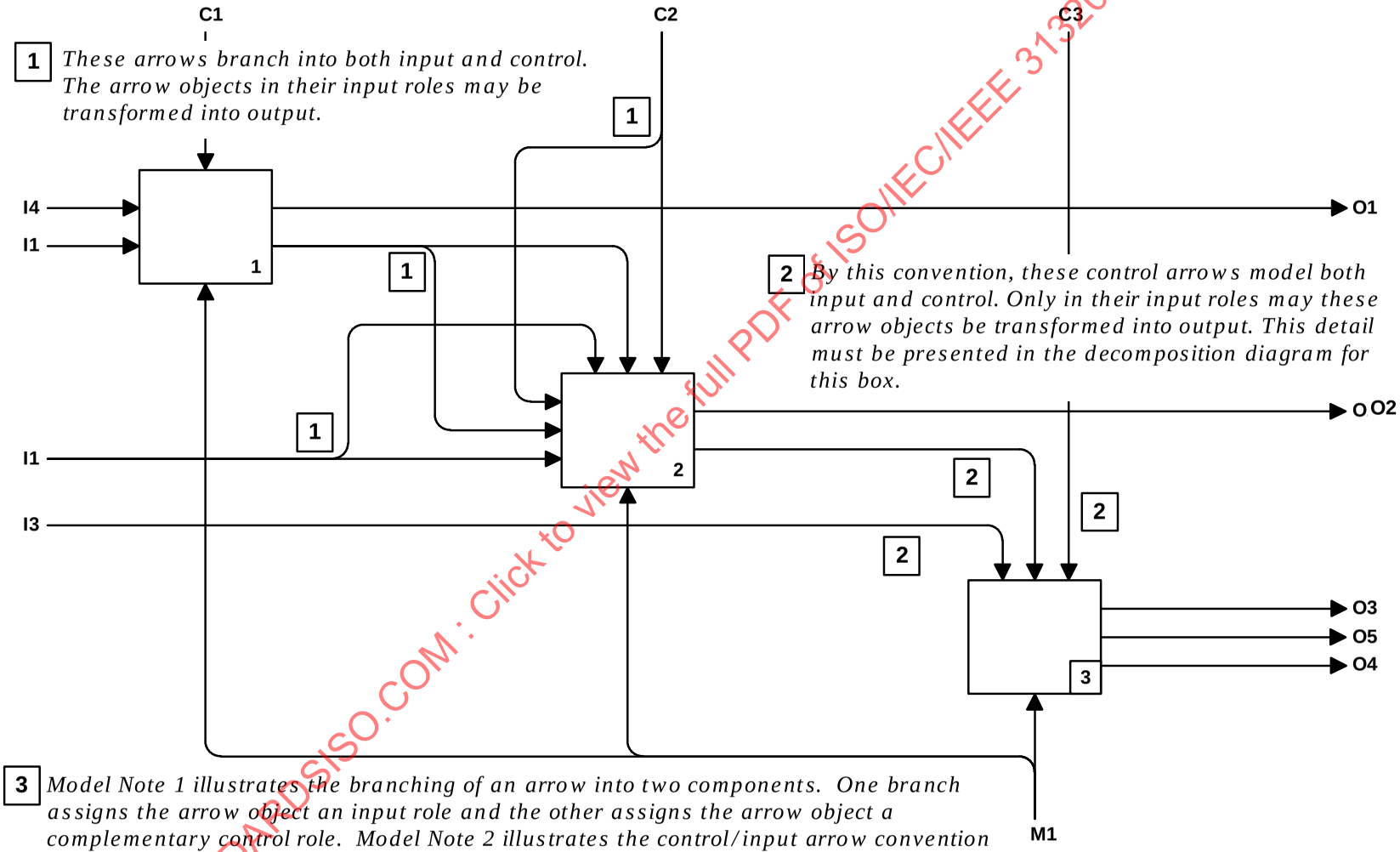
Model Page: FF/S3F4	Figure 9—Label and Name Example (Internal Arrows)	Number: AKB009	P. 66

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:

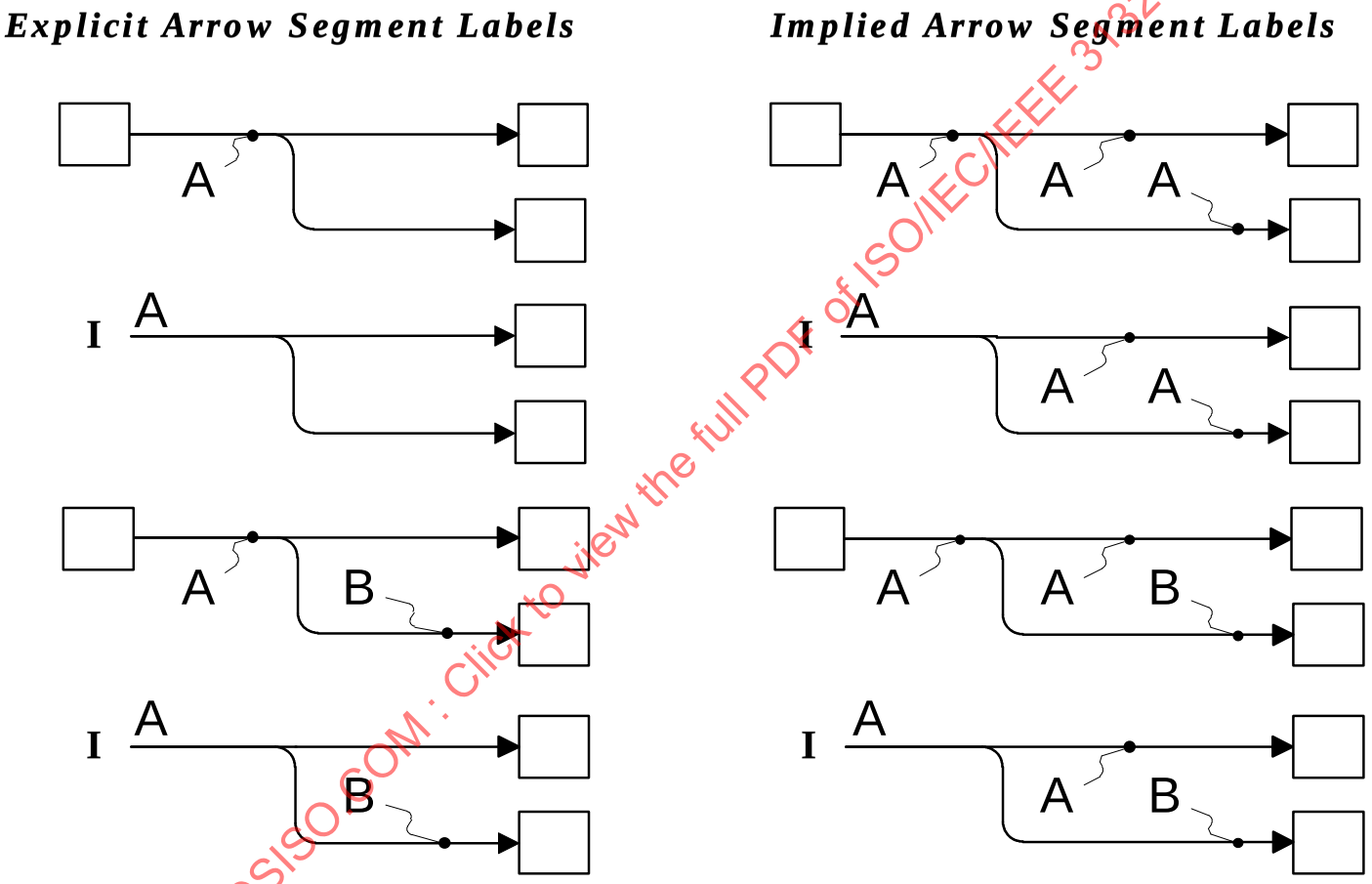


Model Page: FF/S4F1	Figure 10—Arrow Bundling (Meaning)	Number: AKB010	P. 67

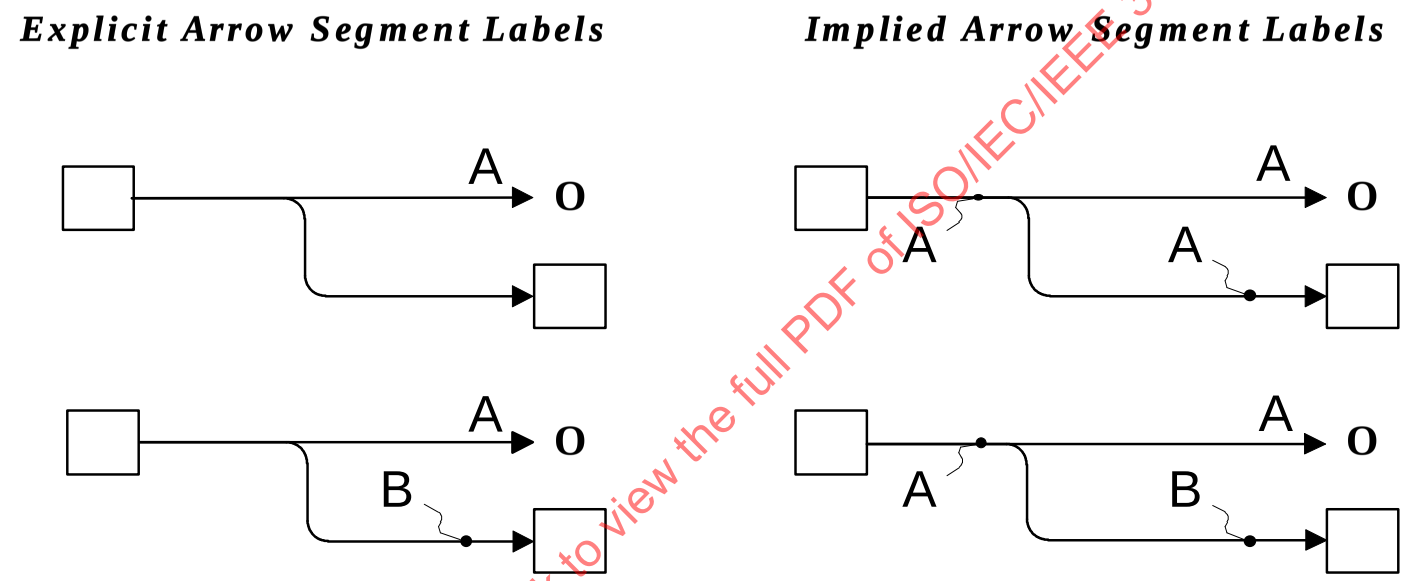
Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



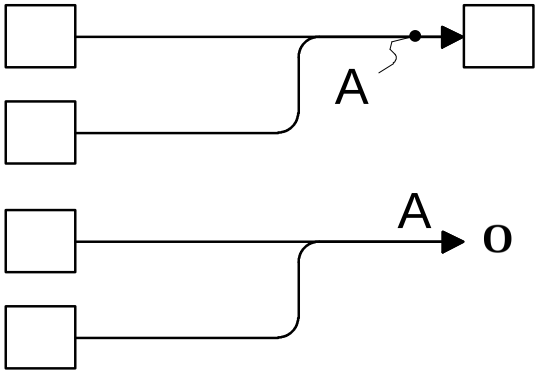
Model Page: FF/S4F2	Figure 11—Arrow Bundling (Roles)	Number: AKB011	P. 68



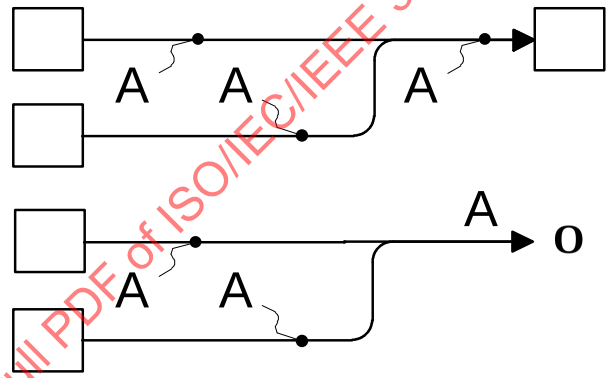
1 *Scope: Holds both for internal arrow segments and for input, control, and mechanism boundary arrows.*



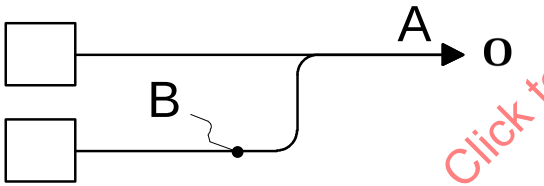
Explicit Arrow Segment Labels

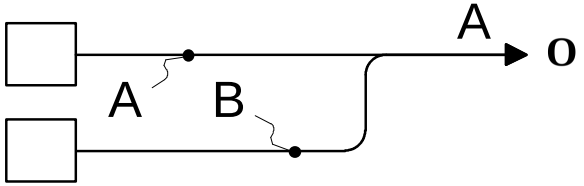


Implied Arrow Segment Labels

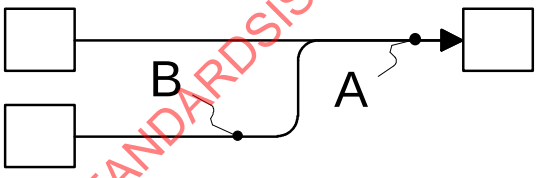


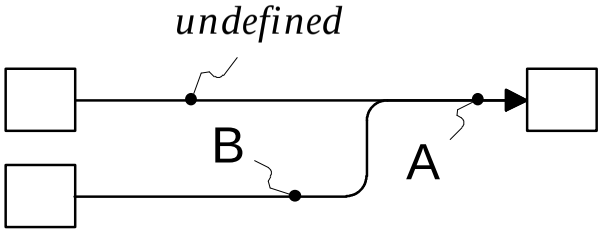
1 *Scope: Holds both for internal arrow segments and for output boundary arrows.*

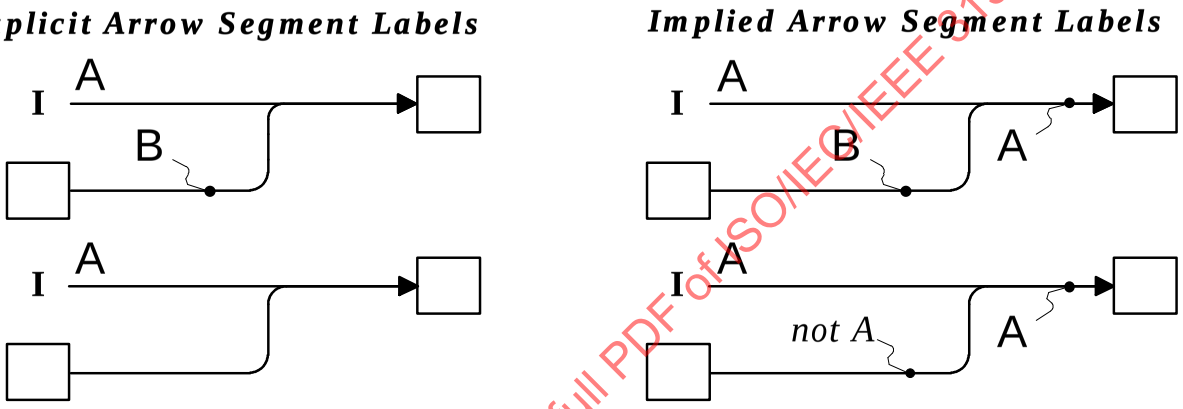




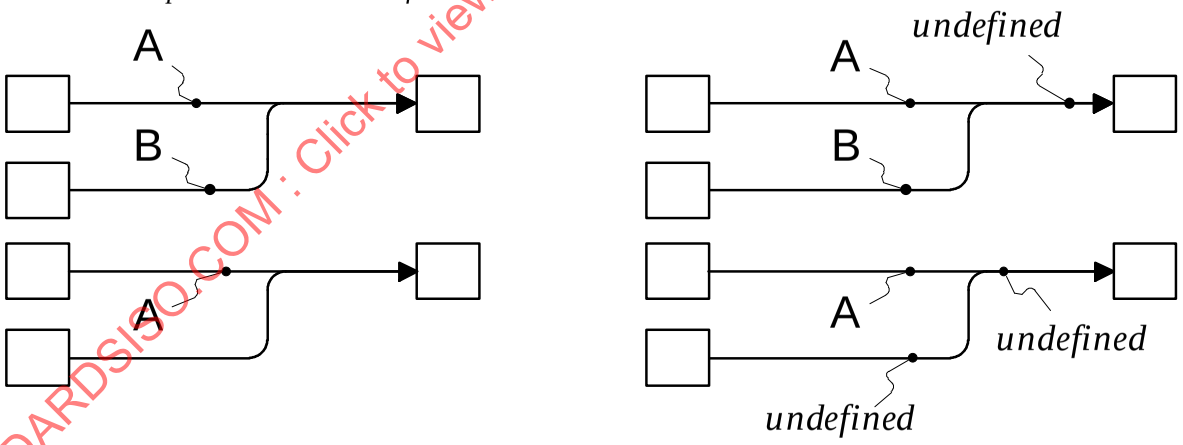
2 *Scope: Holds only for output boundary arrows. Does NOT hold for internal arrow segments. In either case, B must be a subset of A.*



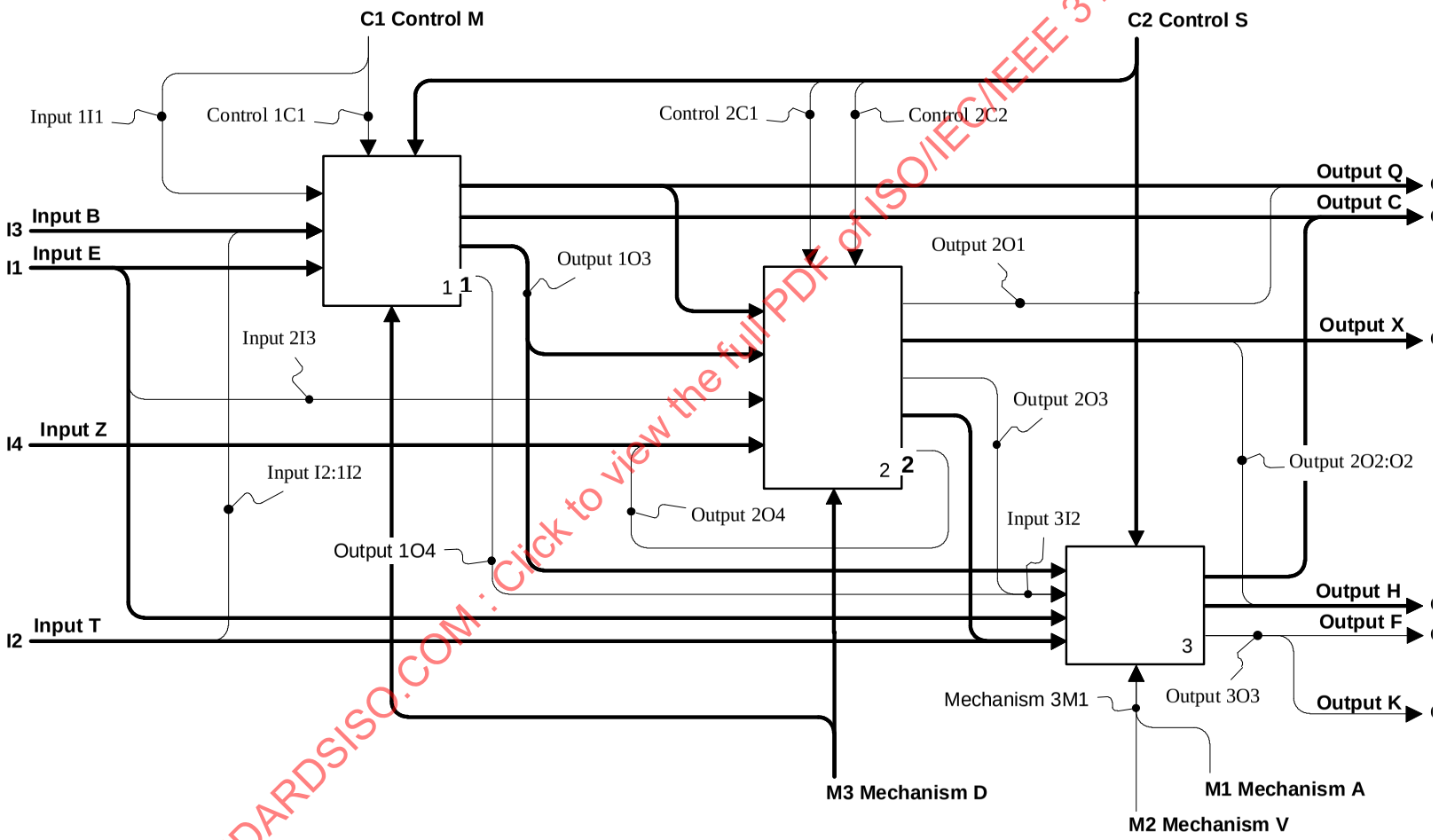




1 *Scope: Holds only for input, control, and mechanism boundary arrows. Missing labels for joining arrow segments are NOT determined by this convention. Does NOT hold for internal arrow segments. In all cases, whether labeled or not, the joining segment must represent a subset of A.*

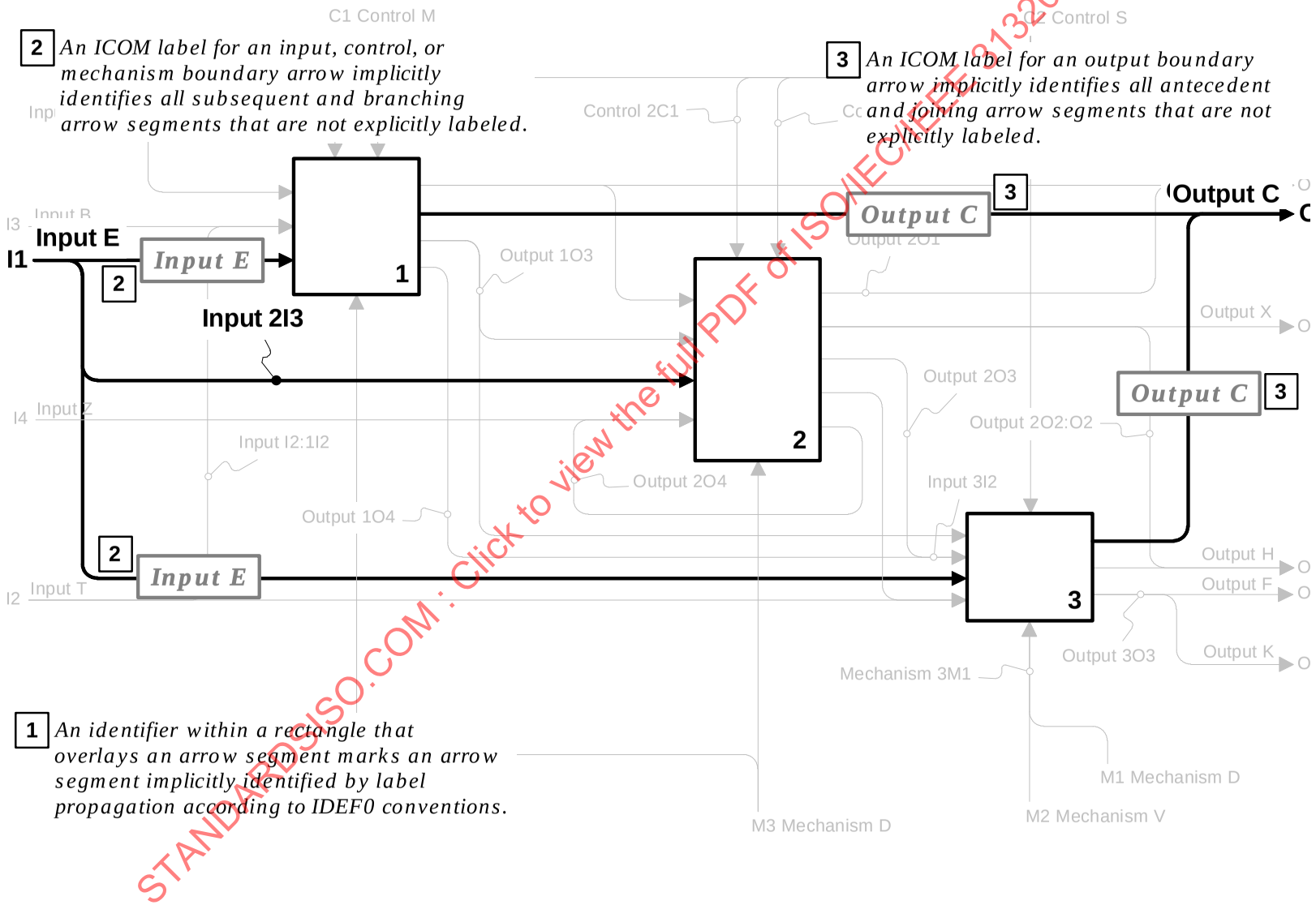


Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



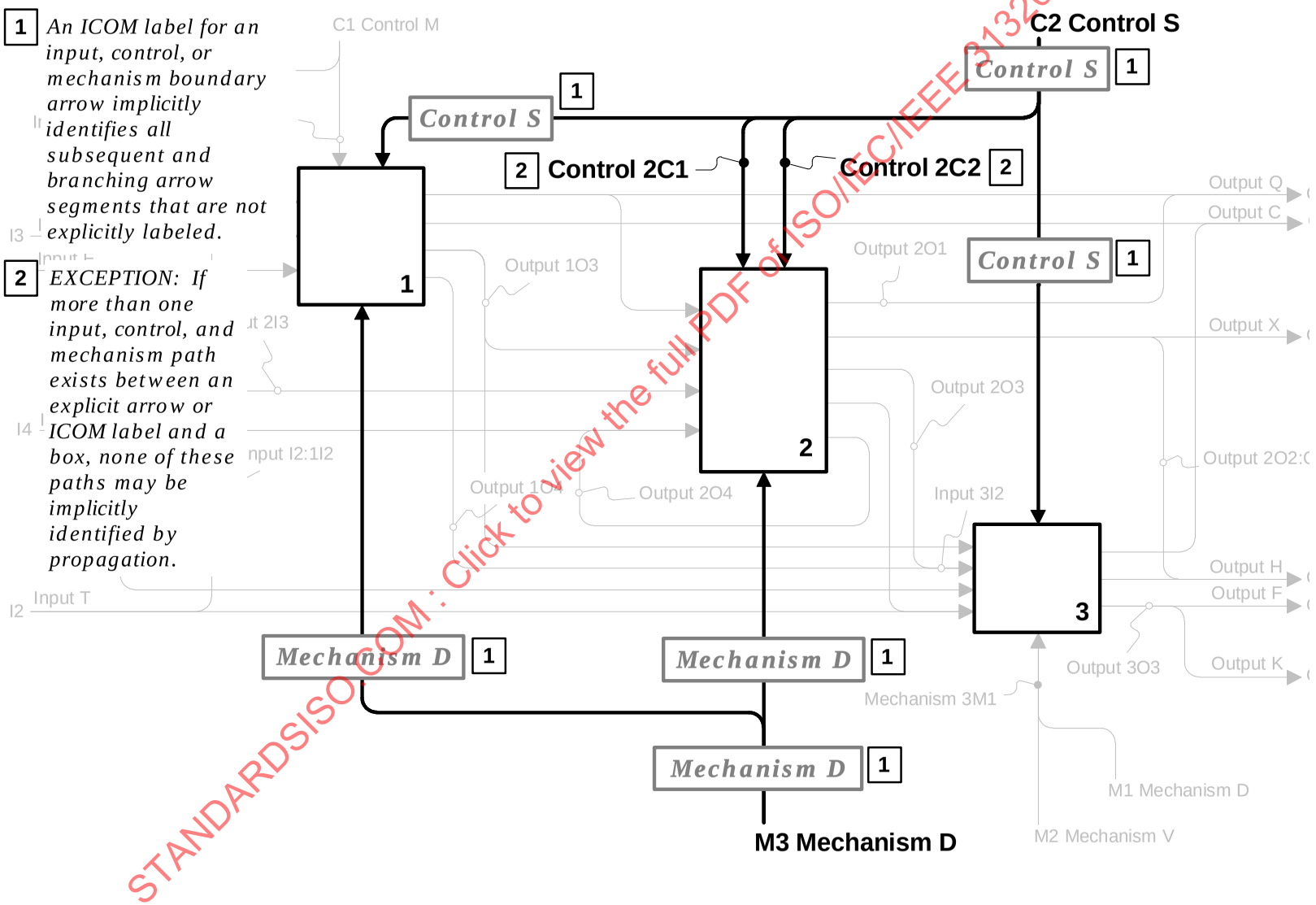
Model Page: FF/S5F5	Figure 16—Label Propagation Examples	Number: AKB016	P. 73

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



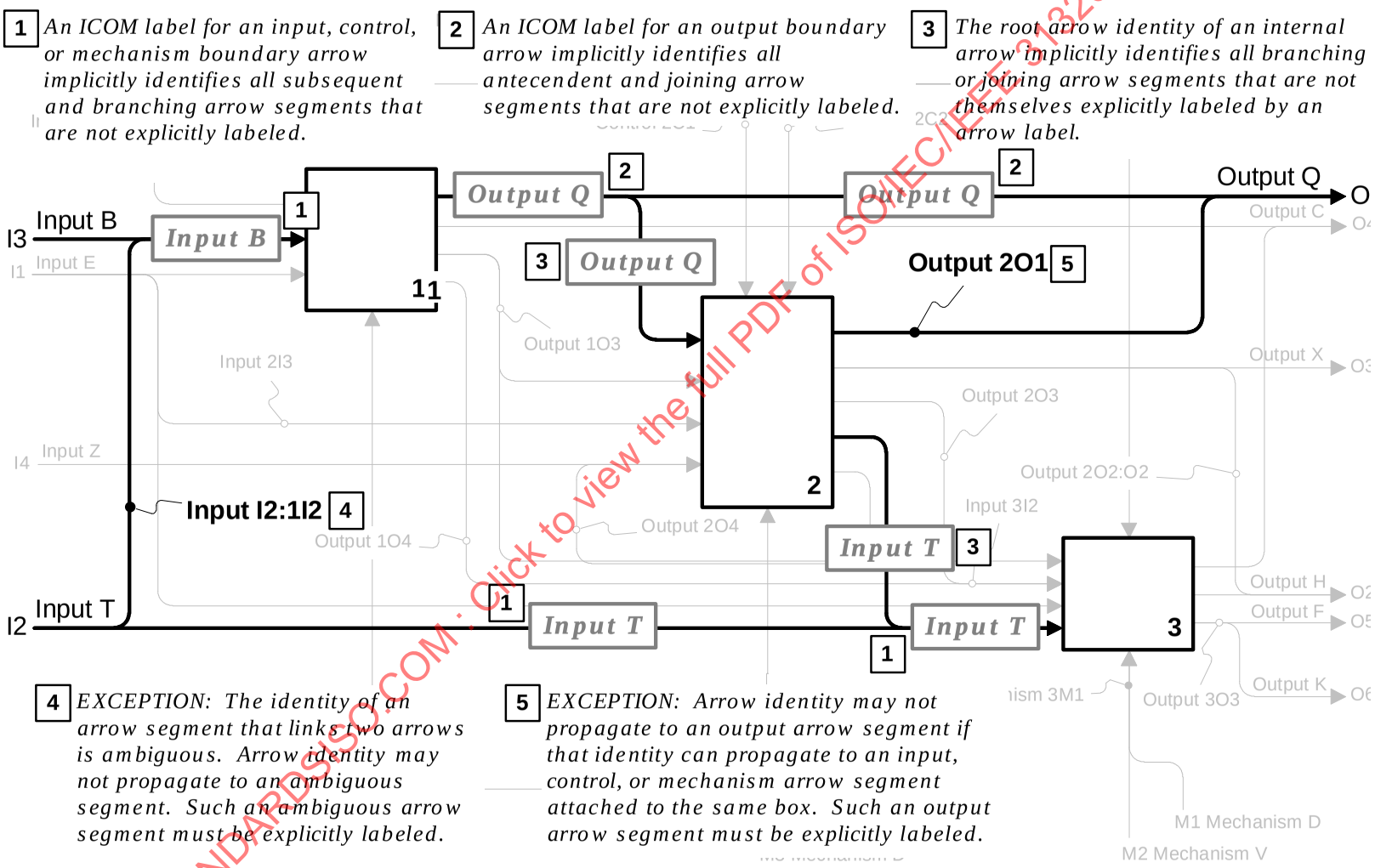
Model Page: FF/S5F6	Figure 17—Label Propagation Example Detail 1	Number: AKB017	P. 74

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



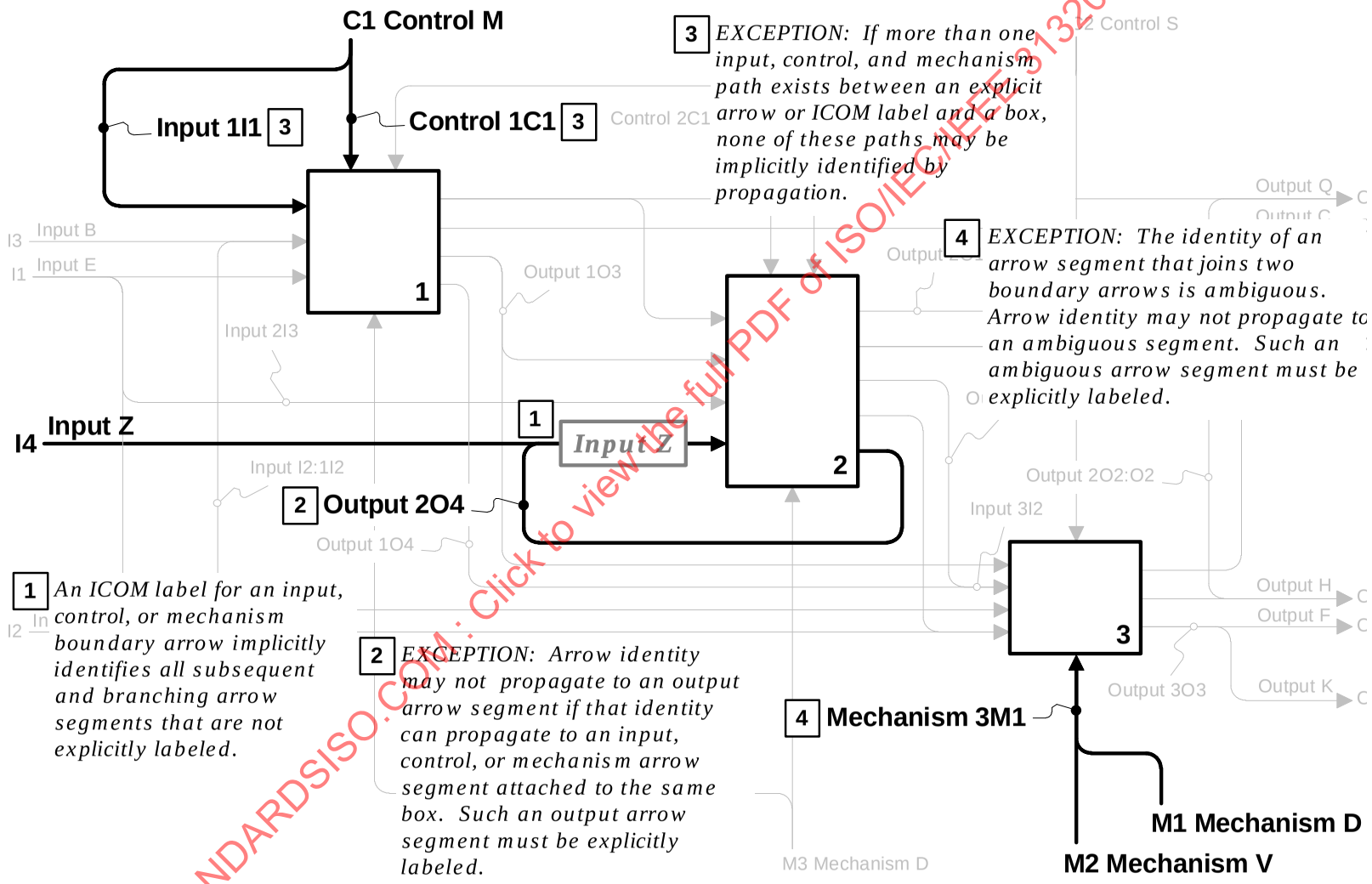
Model Page: FF/S5F7	Figure 18—Label Propagation Example Detail 2	Number: AKB018	P. 75

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



Model Page: FF/S5F8	Figure 19—Label Propagation Example Detail 3	Number: AKB019	P. 76

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:

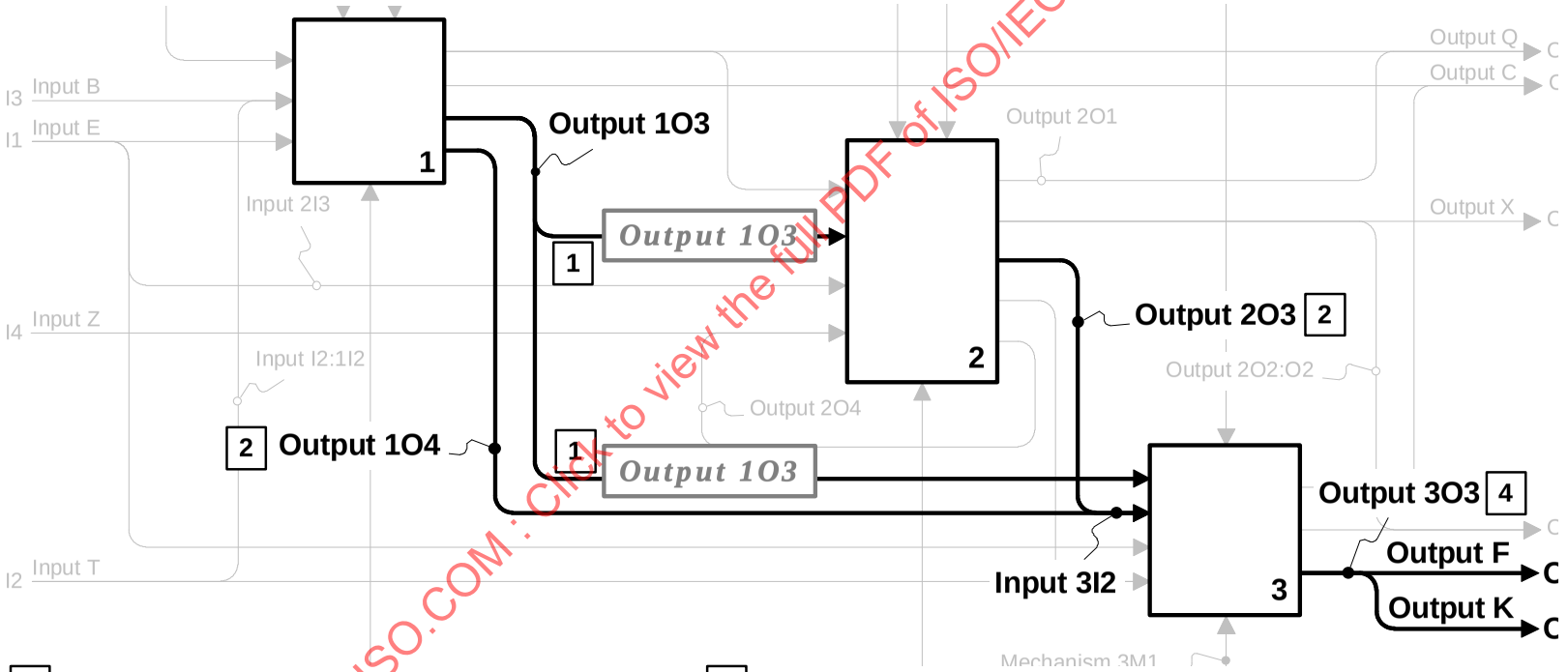


Model Page: FF/S5F9	Figure 20—Label Propagation Example Detail 4	Number: AKB020	P. 77

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:

1 *The root arrow identity of an internal arrow implicitly identifies all branching or joining arrow segments that are not themselves explicitly labeled by an arrow label.*

2 *EXCEPTION: The root arrow identity of an internal arrow implicitly identifies all joining arrow segments that are not themselves explicitly labeled if and only if one and only one joining arrow segment is a constituent of a boundary arrow. Without such a boundary arrow, each joining arrow segment must be explicitly labeled.*

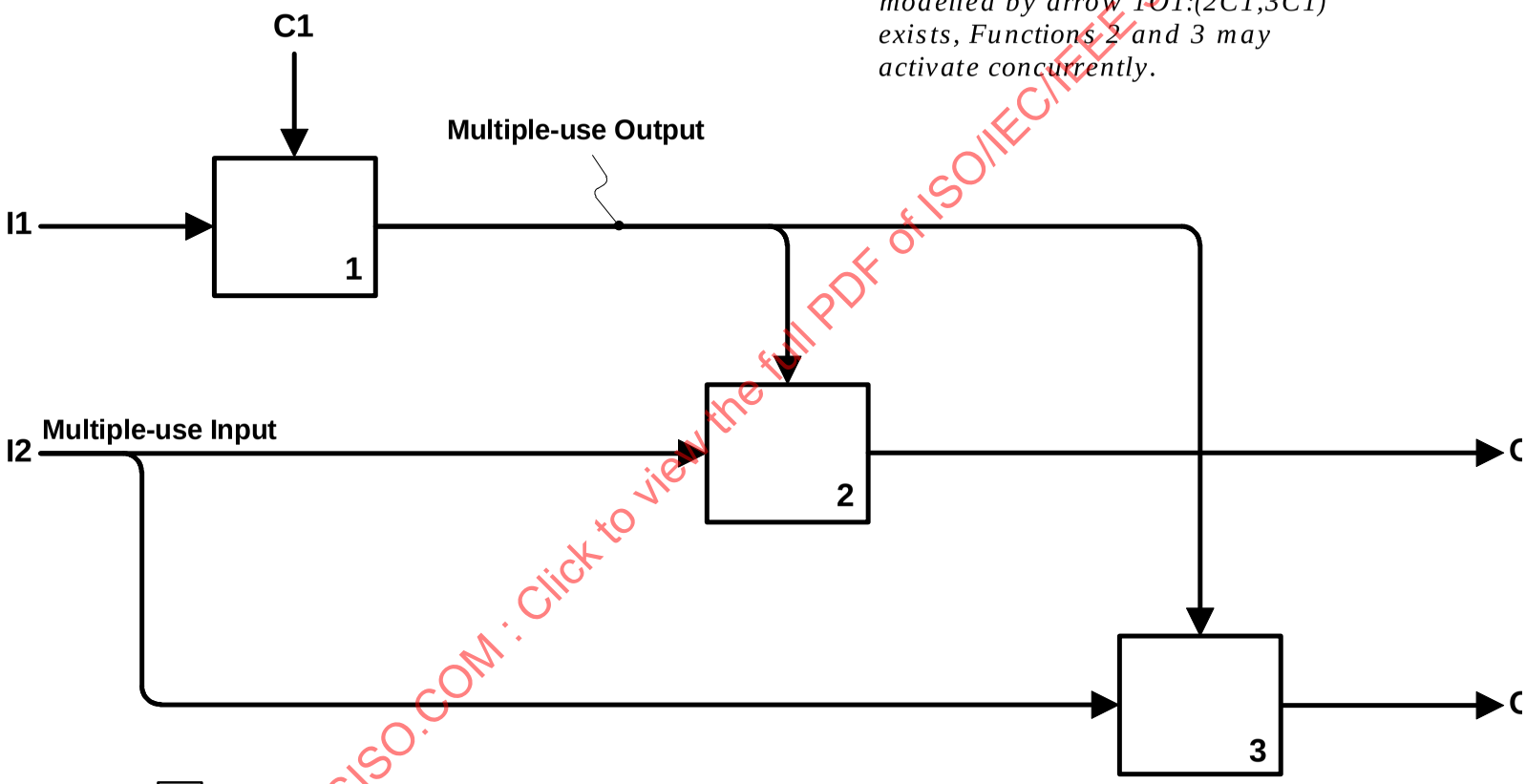


3 *An ICOM label for an output boundary arrow implicitly identifies all antecedent and joining arrow segments that are not explicitly labeled.*

4 *EXCEPTION: The identity of an arrow segment that branches into two boundary arrows is ambiguous. Arrow identity may not propagate to an ambiguous segment. Such an ambiguous arrow segment must be explicitly labeled.*

Model Page: FF/S5F10	Figure 21—Label Propagation Example Detail 5	Number: AKB021	P. 78

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



1 *Once the Multiple-use Output modelled by arrow 1O1:(2C1,3C1) exists, Functions 2 and 3 may activate concurrently.*

2 *Once the Multiple-use Input modelled by arrow 12:(2I1,3I1) exists, Functions 2 and 3 may activate concurrently.*

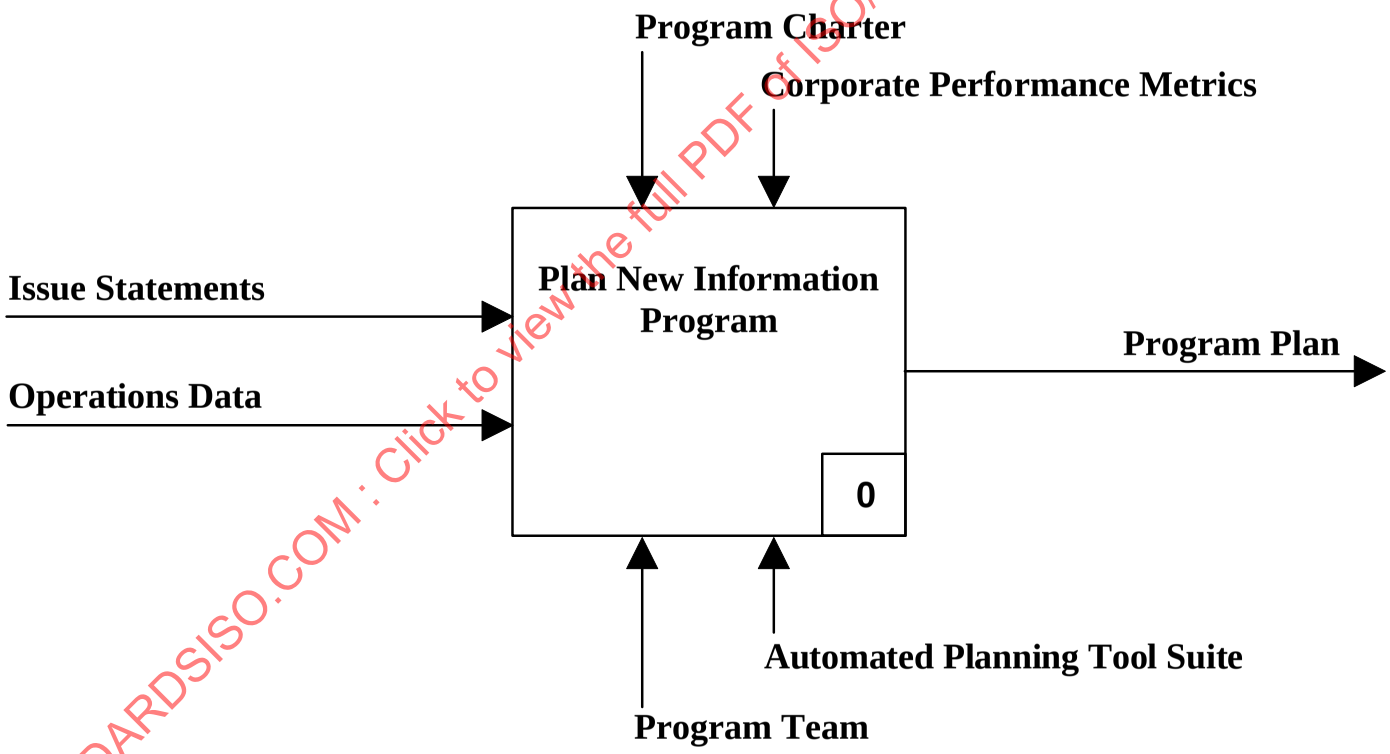
Model Page: FF/S6F1	Figure 22—Concurrent Activation	Number: AKB022	P. 79

Used at:	Author: IEEE IDEF0 Working Group	Date: Oct 1996	X	Publication	Reader Date	Context:
	Project: IDEF0 Language Formalization	Rev: Feb 1998		Recommended		
	Model: Formalization Figures (FF)			Draft		TOP
	Notes: 1 2 3 4 5 6 7 8 9 10			Working		

Model Name: Manage Information Resources (MIR)

Viewpoint: Information Integration Assessment Manager

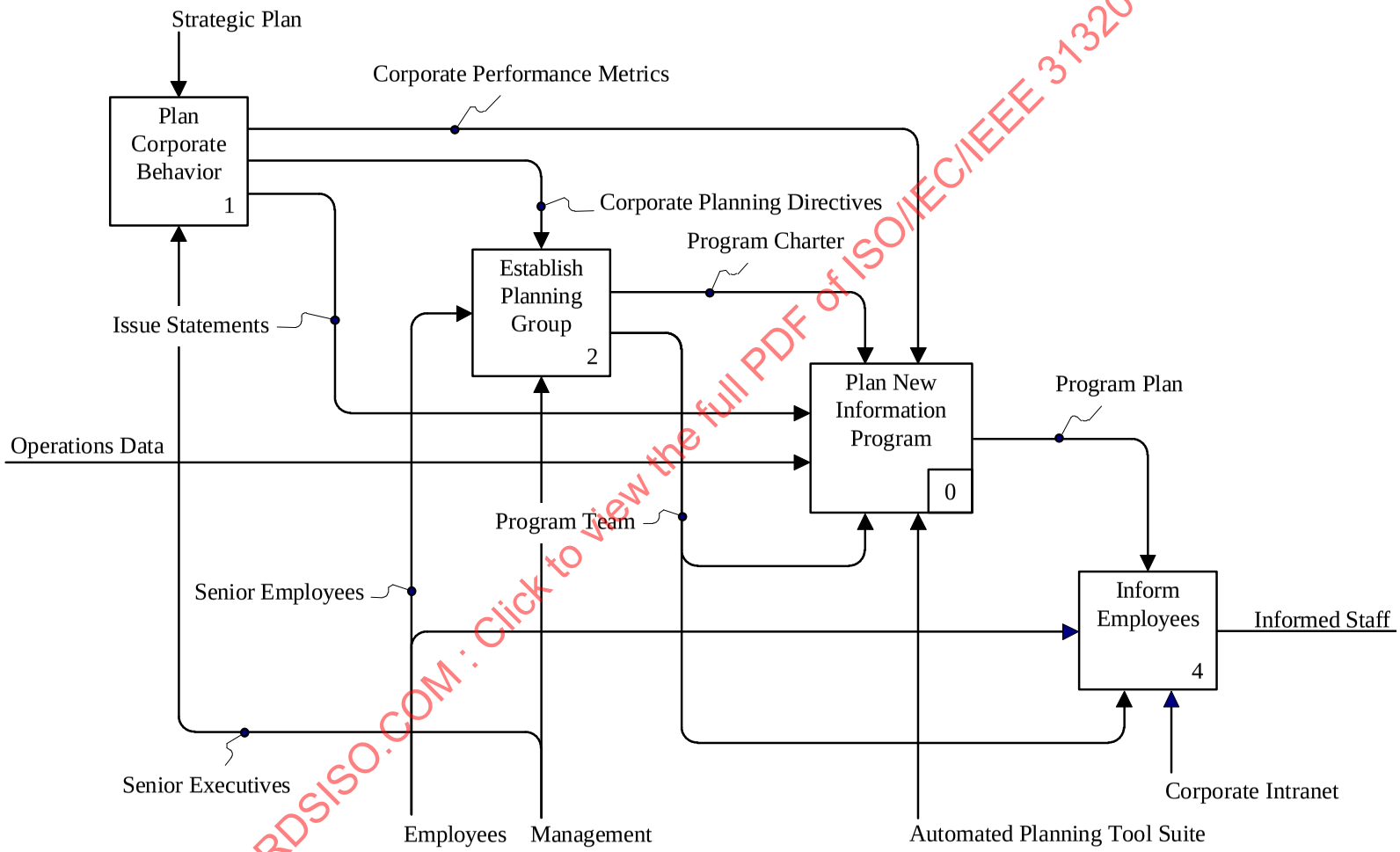
Purpose: To identify the activities required to assess, plan, and streamline information management functions.



Page: MIR/A-0	Title: Context for New Information Program Planning	Number: QRJ12

Model Page: FF/S7F1	Figure 23—Example A-0 Context Diagram	Number: AKB037	P. 80

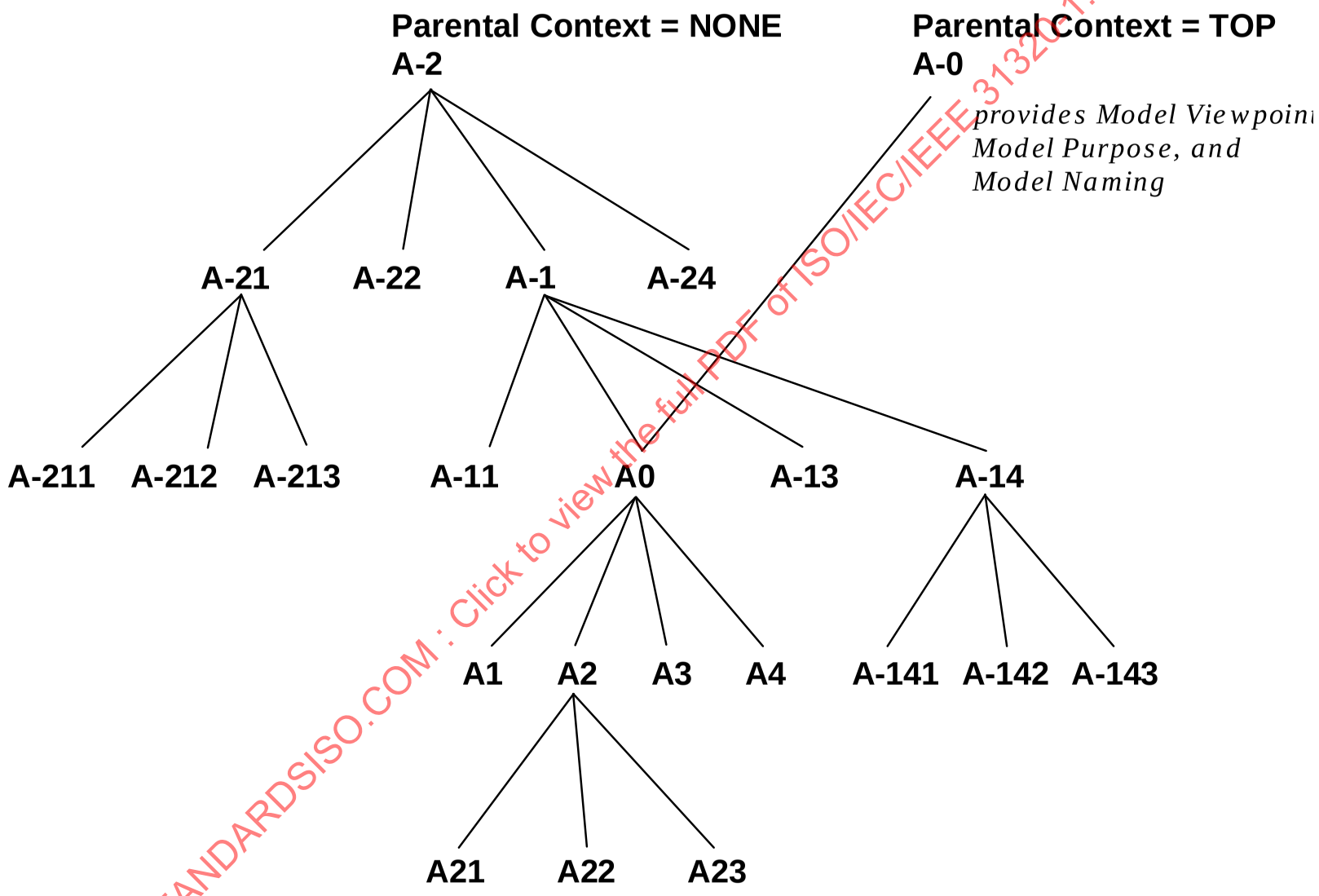
Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context: NONE



Page: MIR/A-1	Title: Context for New Information Program Planning	Number: QRJ16

Model Page: FF/S7F2	Figure 24—Example A-1 Context Diagram	Number: AKB045	P. 81

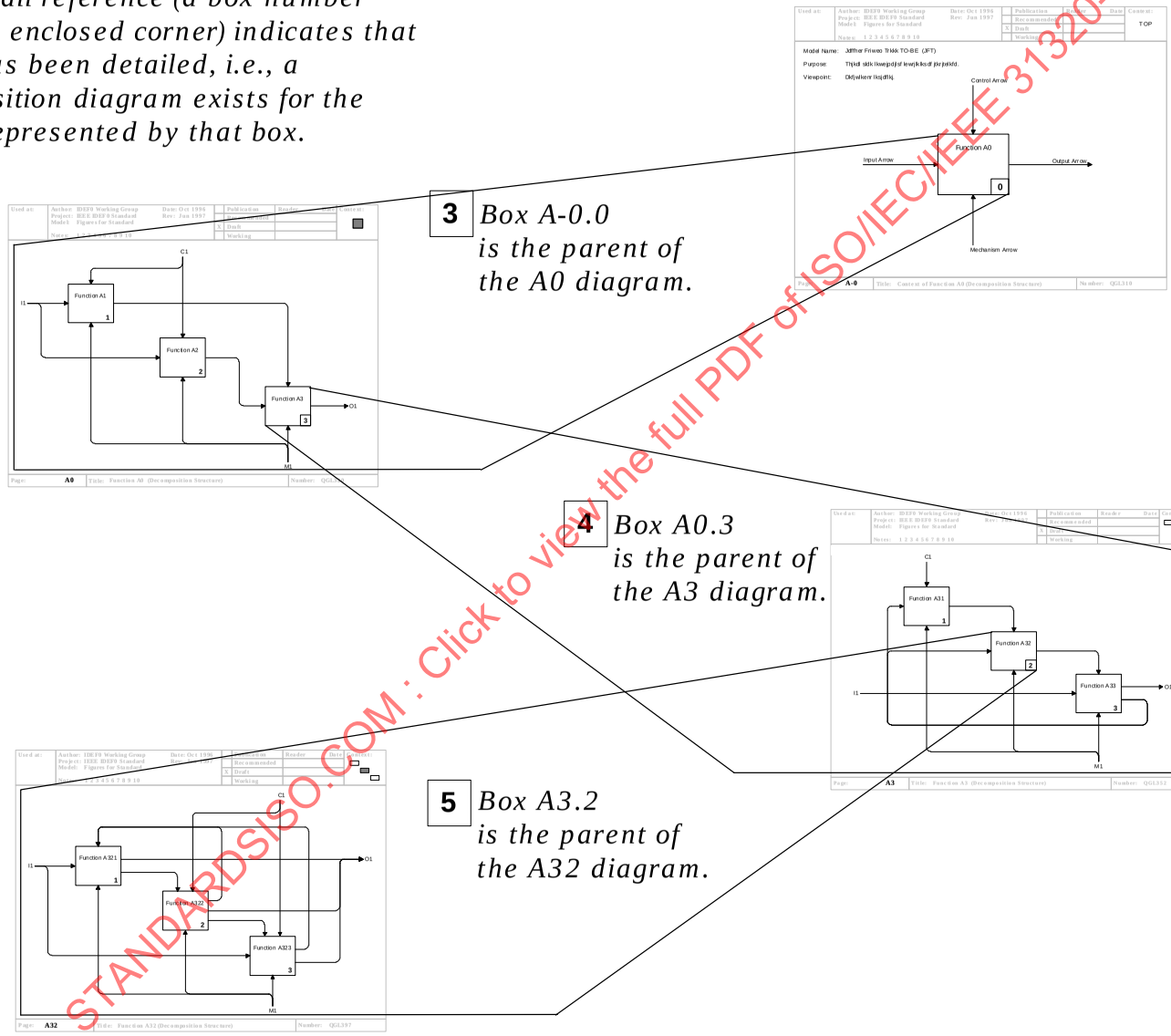
Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



Model Page: FF/S8F1	Figure 25—Example of Context Structure	Number: AKB023	P. 82

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:

2 *A box detail reference (a box number within an enclosed corner) indicates that its box has been detailed, i.e., a decomposition diagram exists for the activity represented by that box.*



3 *Box A-0.0 is the parent of the A0 diagram.*

4 *Box A0.3 is the parent of the A3 diagram.*

5 *Box A3.2 is the parent of the A32 diagram.*

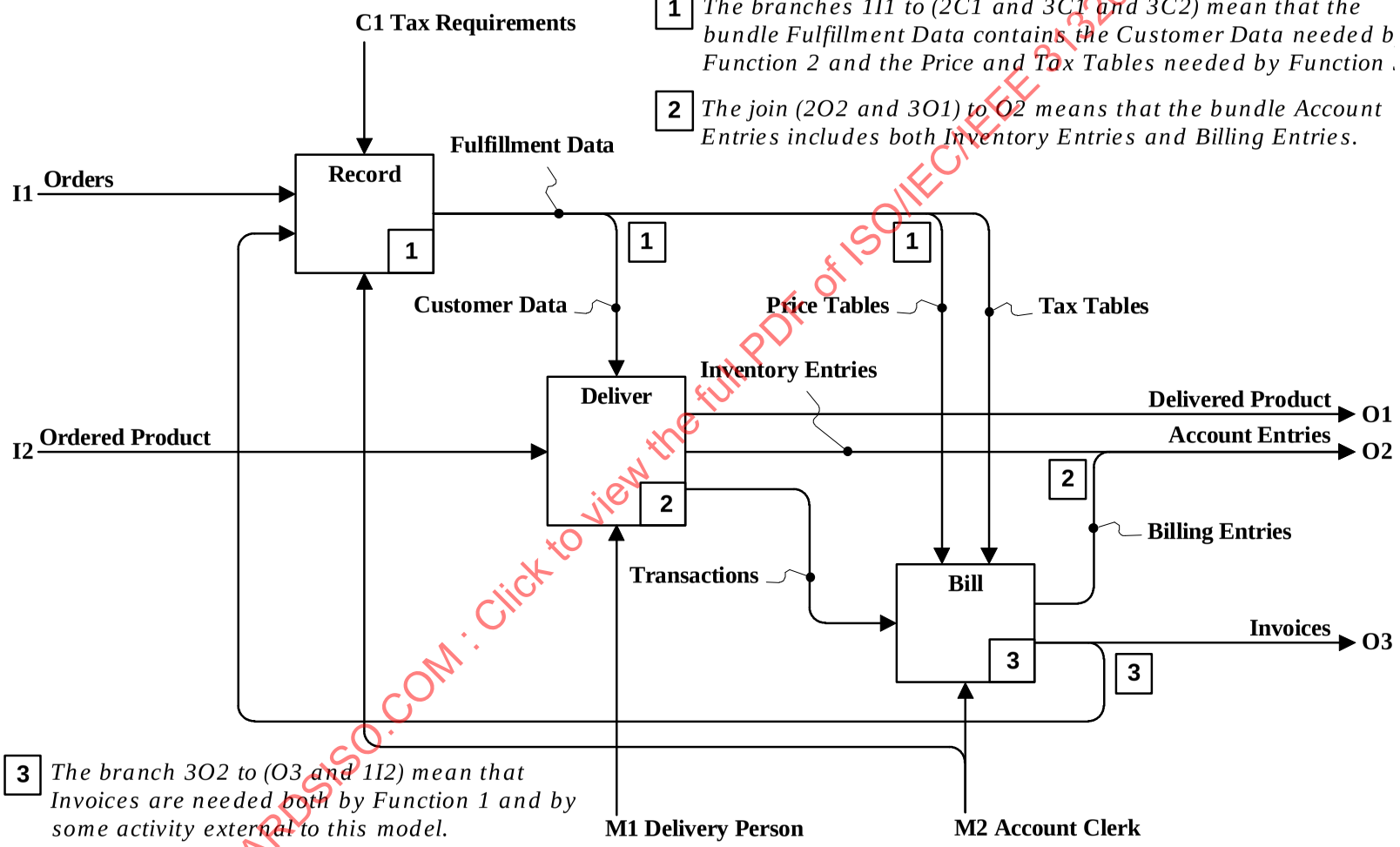
More Genera

1

More Detaile

Model Page: FF/S9F1	Figure 26—Example of Decomposition Structure	Number: AKB024	P. 83

Used at:	Author: IEEE IDEF0 Working Group Project: IDEF0 Language Formalization Model: Formalization Figures (FF) Notes: 1 2 3 4 5 6 7 8 9 10	Date: Oct 1996 Rev: Feb 1998	X	Publication Recommended Draft Working	Reader Date	Context:



Page: A0	Title: Fulfill Orders	Number: MDS008

Model Page: FF/S11F1	Figure 27—Connections among Boxes	Number: AKB038	P. 84