0

AFIT/EN-TR-90-1

Air Force Institute of Technology

An Abstract Data Model for the IDEF$_0$ Graphical Analysis Language

Gerald R. Morris
Capt, USAF

Thomas C. Hartrum

Mark A. Roth
Maj, USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

UNCLASSIFIED

AFIT/EN-TR-90-1

Air Force Institute of Technology

An Abstract Data Model for the IDEF$_0$ Graphical Analysis Language

Gerald R. Morris, Capt, USAF
Thomas C. Hartrum
Mark A. Roth, Maj, USAF

January 11, 1990

Approved for public release; distribution unlimited

An Abstract Data Model for the IDEF$_0$ Graphical Analysis Language

Gerald R. Morris, Thomas C. Hartrum, and Mark A. Roth
Department of Electrical and Computer Engineering (AFIT/ENG)
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-6583

Abstract

IDEF$_0$ is the United States Air Force's ICAM (Integrated Computer Aided Manufacturing) Definition Method Zero graphical analysis language, a subset of Ross' Structured Analysis (SA) language. The language is also an excellent methodology for performing a requirements-phase analysis for any software development project. We present an abstract data model for IDEF$_0$ using entity-relationship diagrams. This model helps to mitigate some of the ambiguities inherent in IDEF$_0$. The model is divided into two parts representing the analysis data (the essential data model) and the graphical data (the drawing data model). This dual modeling approach allows for the extraction of analysis data without having to deal explicitly with the IDEF$_0$ graphical language.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications—*languages, methodologies, tools*; H.2.1 [**Database Management**]: Logical Design—*data models*

General Terms: Languages

Additional Key Words and Phrases: data model, structured analysis, graphical analysis language

1 Introduction

Computer aided software engineering (CASE) represents a set of analysis, design and maintenance methodologies for software and a set of computer-based tools to help automate those methodologies. The Department of Defense is especially interested in how CASE can help reduce the tremendous cost and time needed to develop and maintain very complex software which is embedded in nearly every system under development today. In this article we discuss a particular graphical software analysis language, the United States Air Force's ICAM (Integrated Computer Aided Manufacturing) Definition Method Zero (IDEF$_0$) subset [8] of Ross' Structured Analysis (SA) [9]. ICAM is the Air Force's program for integrated computer aided manufacturing. It is directed towards increasing manufacturing productivity via computer technology. Although the original intent of the IDEF$_0$ language was to provide a structured approach for computer aided manufacturing processes, the language is also an excellent methodology for performing a requirements-phase analysis for any software development project.

One of the problems with both SA and IDEF$_0$ is that there is no formal model of the language. In addition to "blueprint-like graphics," SA and IDEF$_0$ call for the use of natural language [9]. By definition, the use of natural language introduces ambiguity in the overall IDEF$_0$ language. In addition, certain graphical features of IDEF$_0$ allow for ambiguous models to be constructed. One approach to reducing such ambiguity is to replace or augment free-form text with a more syntactically defined data dictionary. Providing a consistent database for a CASE tool, however, requires a data structure that integrates the graphics, text, and data dictionary aspects of the IDEF$_0$ analysis.

In this article, we present an abstract data model for the IDEF$_0$ language. This model helps to mitigate some of the ambiguities inherent in the IDEF$_0$ language by providing a formal specification of the language elements. We take the unique approach of separating the analysis data and the graphical data into separate data models, the *essential* and *drawing* data models, respectively. This dual modeling approach allows for the extraction of analysis data without having to deal explicitly with the IDEF$_0$ graphical language. Thus, the analysis data can be used in an environment that doesn't support the graphical tools of IDEF$_0$ or used with another graphical tool that supports a similar analysis methodology.

The remainder of this article is organized as follows. First, we provide introductions to SA and IDEF$_0$ and how IDEF$_0$ is used at the Air Force Institute of Technology (AFIT). We then discuss briefly the role of database management systems in CASE tools and our adaptation of Chen's entity-relationship model [2]. Finally, we present our abstract data model for IDEF$_0$.

2 Structured Analysis and IDEF$_0$

In his 1976 paper, Douglas Ross introduced Structured Analysis as a generalized language, which allows a complex idea to be represented in a hierarchical, top-down representation [9]. According to Ross, "The human mind can accommodate any amount of complexity as long as it is presented in easy-to-grasp chunks that are structured together to make the whole" [9:17]. SA combines graphic features such as lines and boxes with standard written language to create the SA model. Figure 1 illustrates the basic idea behind this

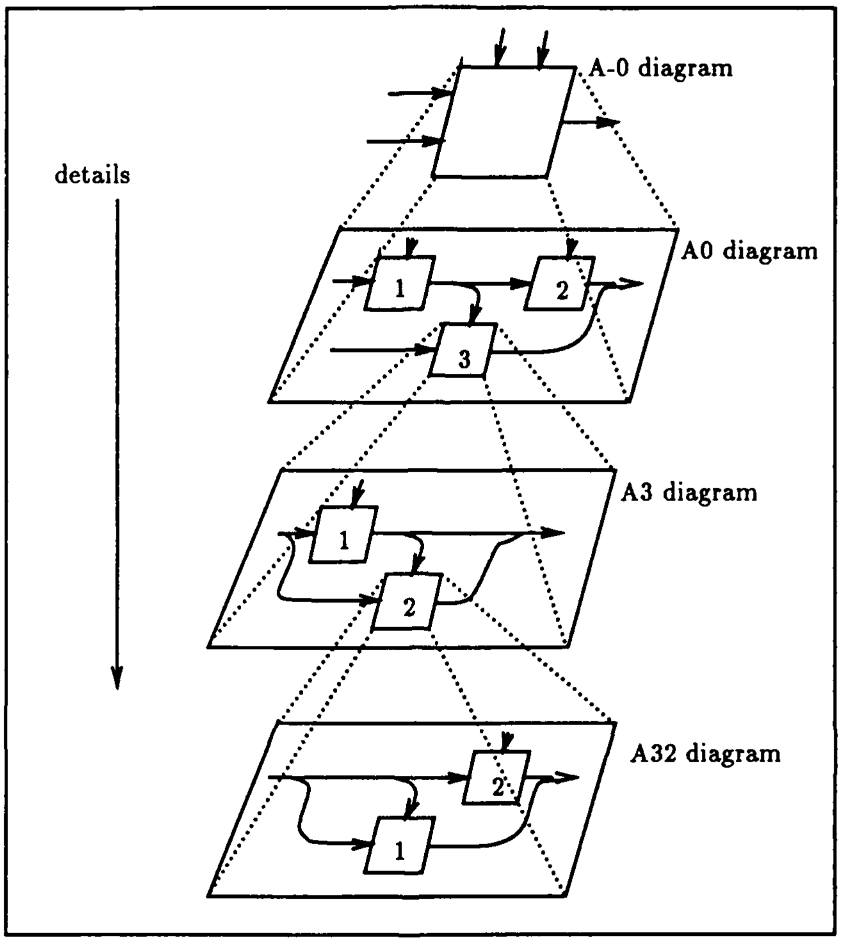

Figure 1: Structured Decomposition (Based on [9:18])

structured decomposition. At each level, only the details essential for that level are given. Further details are exposed by moving down in the hierarchy.

SA provides for two kinds of decomposition, an activity decomposition, and a data decomposition. In the activity decomposition, activities (verbs) are represented by rectangular boxes, and the data (nouns) are represented by arrows flowing into and out of the boxes. In the data decomposition, the boxes represent data (nouns), and the arrows represent activities operating on the data in the boxes. An example of an activity decomposition is shown in the following two figures. Figure 2 represents the overall context of the system being analyzed (referred to as the "A minus zero" diagram).

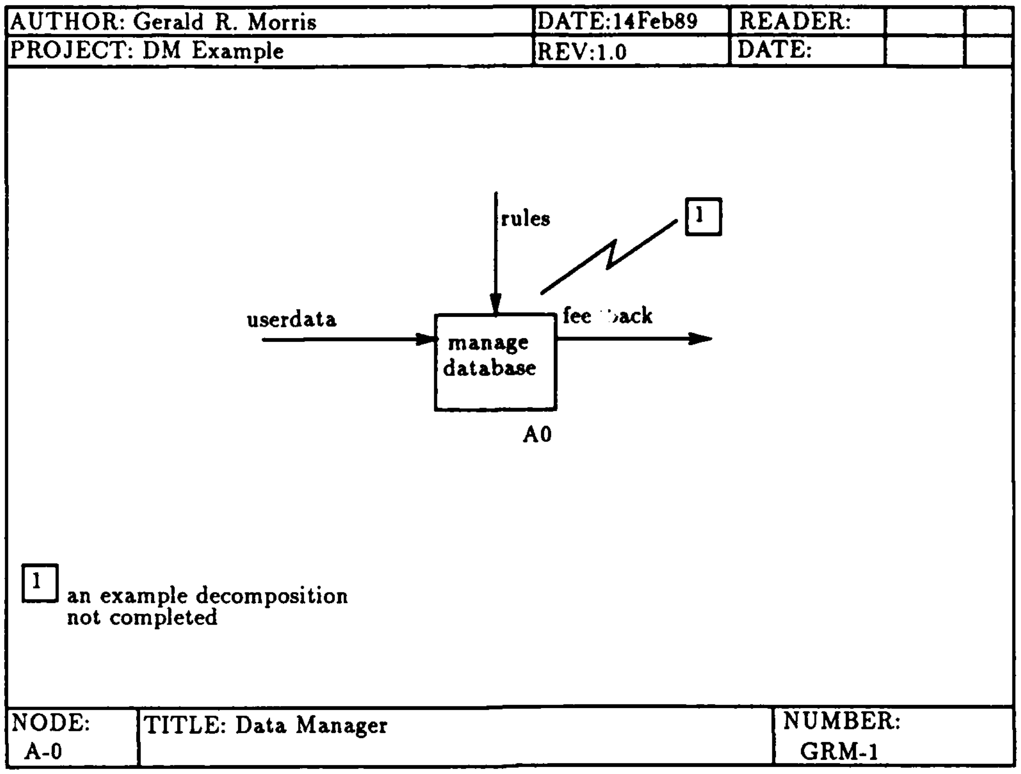

Figure 2: A-0 Diagram

Figure 3 represents the first level decomposition. In a real analysis, the A0 diagram would be further decomposed to whatever level was necessary to ensure an unambiguous interpretation of the system requirements. Marca and McGowan have written an excellent book which describes SADT[1] and provides numerous workshop-style examples with which users can develop a flavor for the language [7].

A full implementation of SA includes 40 different language features, and the dual decomposition [9:20]. But the United States Air Force Program for Integrated Computer Aided Manufacturing (ICAM), which is directed towards increasing manufacturing productivity via computer technology, defined a subset of Ross' Structured Analysis language called ICAM Definition Method Zero (IDEF$_0$) [8]. This functional modeling language eliminates some of the more esoteric features of Ross' language, as well as the data decomposition. According to the IDEF$_0$ manual

[1]Structured Analysis and Design Technique (SADT) is SofTech's name for SA and is a registered trademark

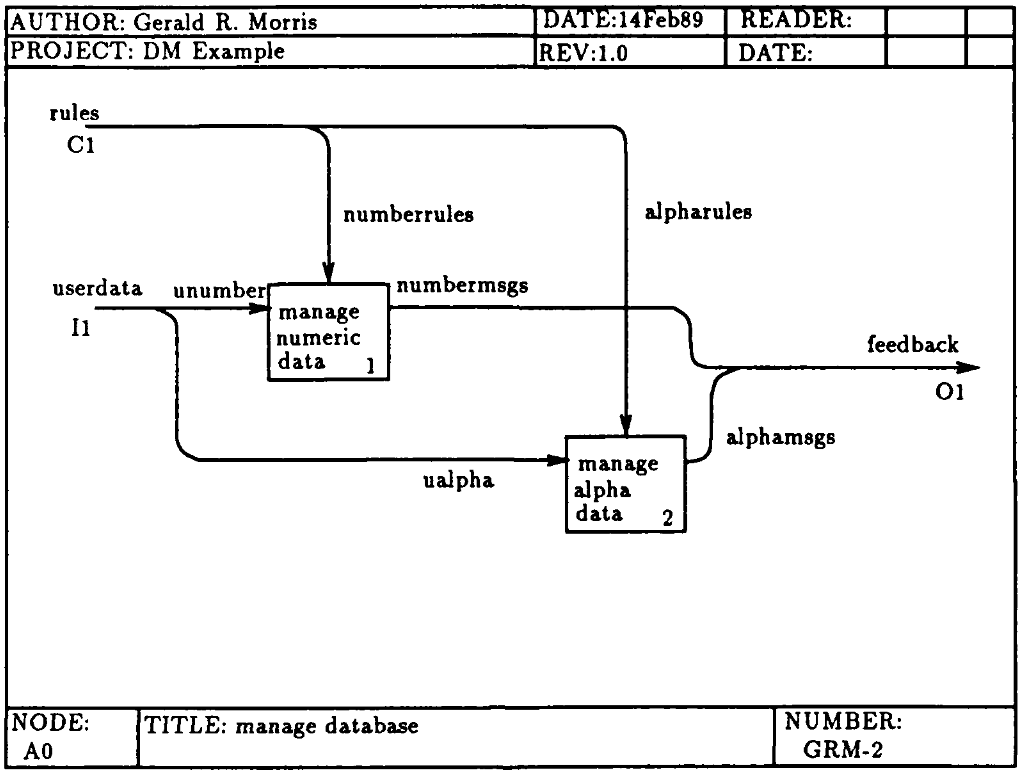

Figure 3: A0 Diagram

> The ICAM program approach is to develop structured methods for applying computer technology to manufacturing and to use those methods to better understand how best to improve manufacturing productivity. ...IDEF$_0$ is used to produce a *function* model which is a structured representation of the functions of a manufacturing system or environment, and of the information and objects which interrelate those functions. [8:1–1]

One of the problems with both SA and IDEF$_0$ is that there is no formal model of the language. In addition to "blueprint-like graphics," SA and IDEF$_0$ call for the use of natural language [9]. The use of such natural language, by definition, introduces ambiguity in the overall IDEF$_0$ language. In addition, certain graphical features of IDEF$_0$ allow for ambiguous models to be constructed. In short, IDEF$_0$ is not a rigorous language.

Obviously, the original intent of the IDEF$_0$ language was to provide a structured approach for computerizing manufacturing processes. However, as a subset of SA, the language also provides an excellent methodology for performing a requirements-phase analysis for software development projects. It is precisely this use of IDEF$_0$ which was the motivation for developing the model presented in this article.

3 AFIT and IDEF$_0$

At the Air Force Institute of Technology (AFIT) Department of Electrical and Computer Engineering we have promulgated a set of system development guidelines and standards which encourage consistency throughout all phases of hardware and software systems development [5]. As part of this standard, and in conjunction with ongoing efforts to utilize computer aided software engineering (CASE) in our software engineering curriculum, we have selected IDEF$_0$ for performing systems analyses. We have extended the IDEF$_0$ language to include a data dictionary, which is AFIT's implementation of and improvement over the glossary called for by IDEF$_0$. It provides not only the glossary, but also a more syntactical representation of some of the ambiguous features of the language. Several CASE tools have been developed to assist AFIT students during software development. Of particular interest is a tool called SAtool, based on the IDEF$_0$ language, which allows a software engineer to perform an analysis of software requirements [6].

SAtool, which runs on a Sun workstation, is a graphics based editor which allows the analyst to draw the diagrams and enter portions of the data dictionary for the requirements analysis phase of software development. The remaining elements of the data dictionary are automatically derived from the diagram. The user can generate a printout of the SA diagram, a so-called *facing-page text* printout, and a hard-copy printout of the data dictionary. The analysis results can be saved in a standard data file for uploading into a common database. The tool also saves the graphical drawing information so the user can recall the diagram for editing. Figure 4 illustrates some of the SAtool products.

The standard data file generated by SAtool can be uploaded into AFIT's common database. According to Connally, the goal of the common database system is to

> provide an integrated system in which a designer could sit down at a workstation, download the necessary data from a central database, work on a portion of the design, and when finished, upload the data back to the database. [3:2]

Many CASE tool vendors, however, do not use database technology in the development of their products. Typically they create some flat file to save all the information generated by the tool. In other tools even though a DBMS is provided, a direct interface to the database is not provided.

The AFIT SAtool is no exception. In order to save the SAtool data in the AFIT common database, the tool is forced to "walk" the diagram and extract the required information. The user then uploads this standard data file onto the central system which contains the common database and, via Connally's interface,

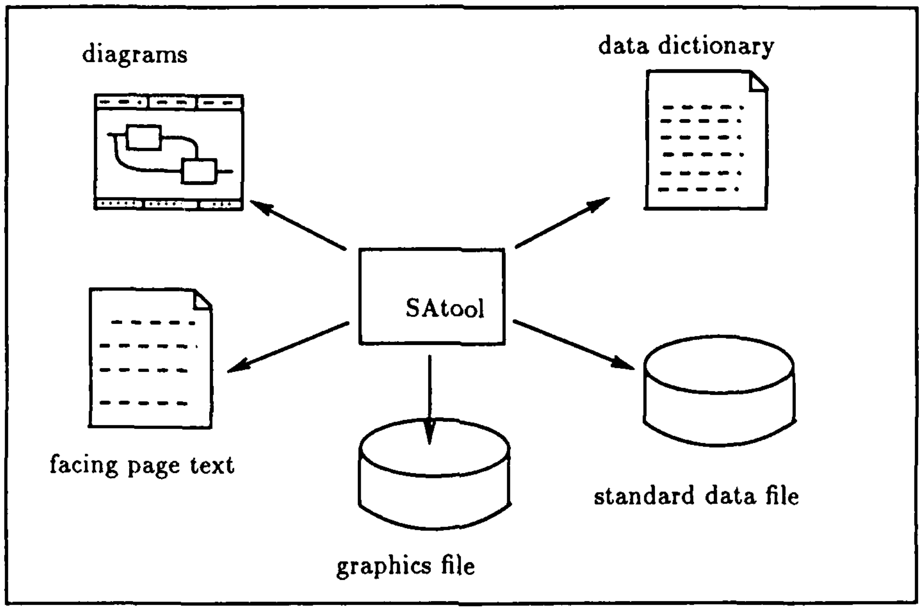

Figure 4: SAtool Products

loads the data into the database. This creates obvious problems relative to sharing the data generated by the tool with other tools since every tool would be required to provide, or convert to, the standard data file format. One potential solution is to use some type of database management system (DBMS) in conjunction with the CASE tool, as opposed to the file conversion scheme. The development of an abstract data model for the $IDEF_0$ language is our first step in the development of such a DBMS and in providing a blue-print for the internal data structures of SAtool.

Our model is based on Chen's entity-relationship (E–R) model [2]. Basically the E-R model depicts the concepts of entities, relationships between entities, and attributes of entities and relationships. An E-R diagram based upon an example in Date [4] is shown in Figure 5. We have adapted Chen's E-R methodology to make things more understandable. In particular, we have added a line on the side of the relationship which clarifies how it relates to the corresponding entities. For example, Figure 5 would be read as "supplier supplies parts." Additionally, we have made the cardinality obvious, "supplier supplies zero to many parts," and "parts are supplied by one to many suppliers." Finally, we have added an asterisk on the attribute which serves as the key, e.g., *s#* is the key attribute for entity, **supplier**.

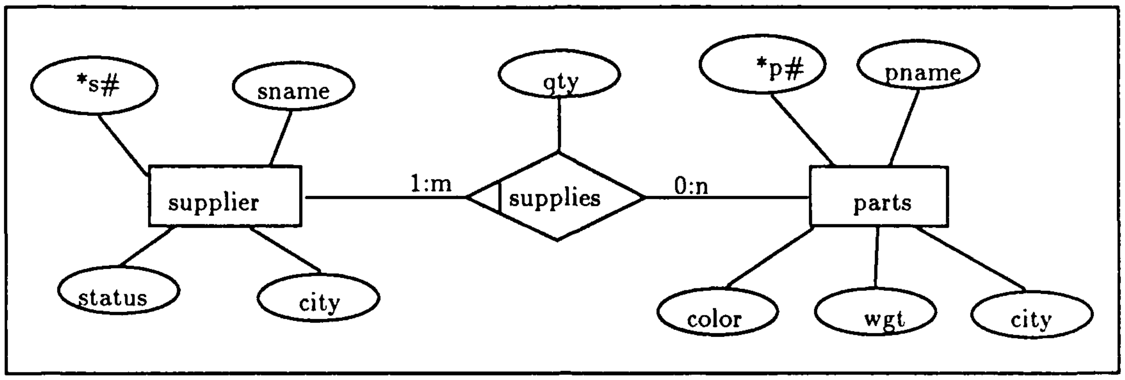

Figure 5: Modified Entity-Relationship Notation

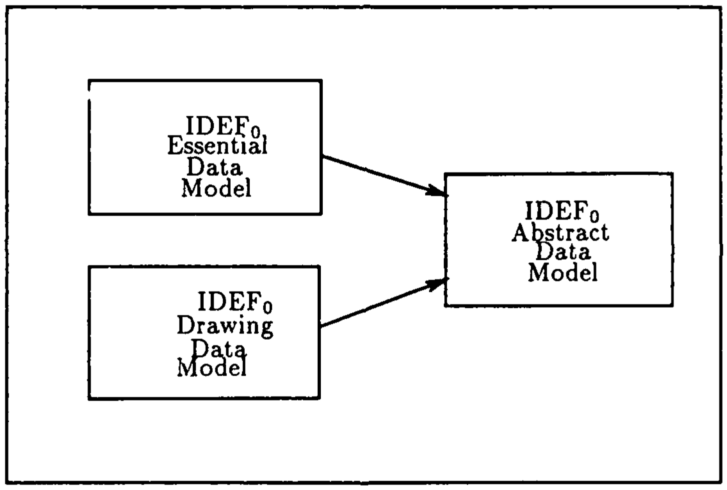

Figure 6: IDEF$_0$ Abstract Data Model: Essential Data and Drawing Data

4 Development of an Abstract Data Model

In order to facilitate development of a DBMS implementation of the IDEF$_0$ language, an abstract data model of the language was constructed. This abstract data model consists of two parts; (1) the essential data model, and (2) the drawing data model. The concept is illustrated in Figure 6. This dual modeling approach allows for the extraction of analysis data without having to deal explicitly with the IDEF$_0$ language, i.e., without having to "walk through" the various drawings.

The IDEF$_0$ essential data model captures those portions of the language which represent the semantics of a particular analysis. A given IDEF$_0$ analysis could actually be represented by infinitely many drawings.

These could differ by as little as the position of a box on the diagram or by the use of optional IDEF$_0$ shorthand graphics symbols such as two related feedback arrows vs. a double-headed arrow. This is similar to representing a linear system by a list of equations or by a coefficient matrix. Both forms are representations of the same underlying model. In a similar sense, an IDEF$_0$ analysis may be syntactically expressed any number of ways, yet still convey the same underlying semantical information. It is this underlying "essential" data which is being captured by the essential data model[2]. This includes, for example, activities and their children, as well as data elements. It does not include, for example, the location of the boxes or arrows which graphically represent the activities and data elements.

The drawing data model represents the graphical constructs used to represent the particular IDEF$_0$ analysis. It contains such information as the location of boxes, the line segments which constitute a given data element, various graphics artifacts, such as the location of "squiggles," the location of footnote markers, some of the graphics "short-hand" such as double headed arrows, etc. This drawing data, in conjunction with the essential data, is used to actually draw an IDEF$_0$ diagram.

The entity-relationship method is used to analyze both the essential data model and the drawing data model since it retains many of the semantics of the actual data being modeled.

4.1 Essential Data Model

In order to allow for an understandable, yet complete representation, the E-R analysis of the essential data model is done in two parts that complement one another. The first portion of the E-R analysis shows the activity model, with the details about data elements left out. The second portion shows the data element model while leaving out the details about the activities.

Figure 7 illustrates the essential model associated with IDEF$_0$ activities and Figure 8 illustrates the essential model associated with IDEF$_0$ data elements. Each of the entities and relationships for both E-R diagrams is explained in Table 1. Most of the attributes include a reference to show why the given attribute was necessary.

[2]This is similar to the concept of an "essential system model" as suggested by Yourdan [10] as defining *what* the system does and not *how* it is implemented. Our essential model is also independent of how it is (graphically) represented.

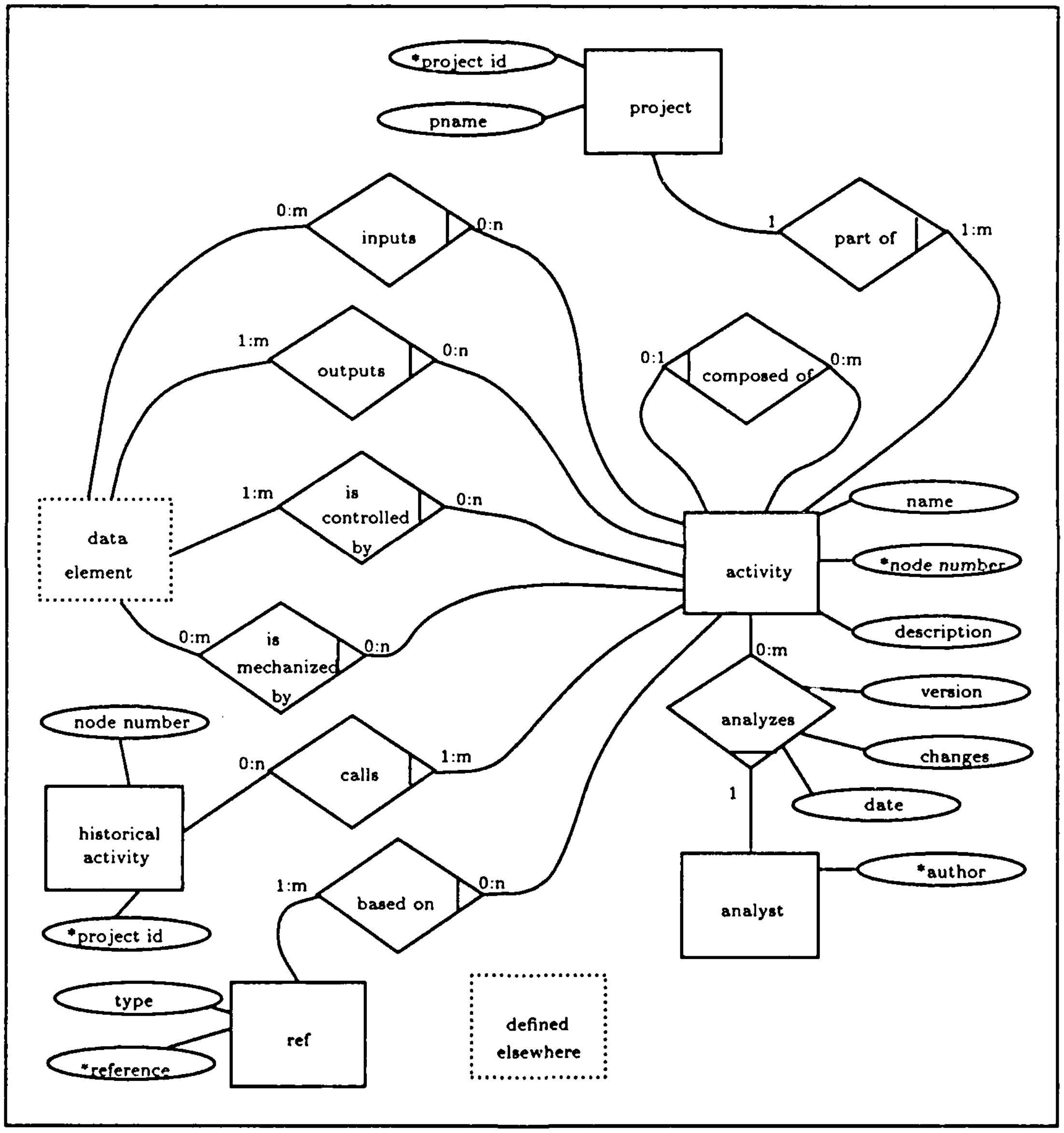

Figure 7: IDEF$_0$ ACTIVITY Essential Data Model

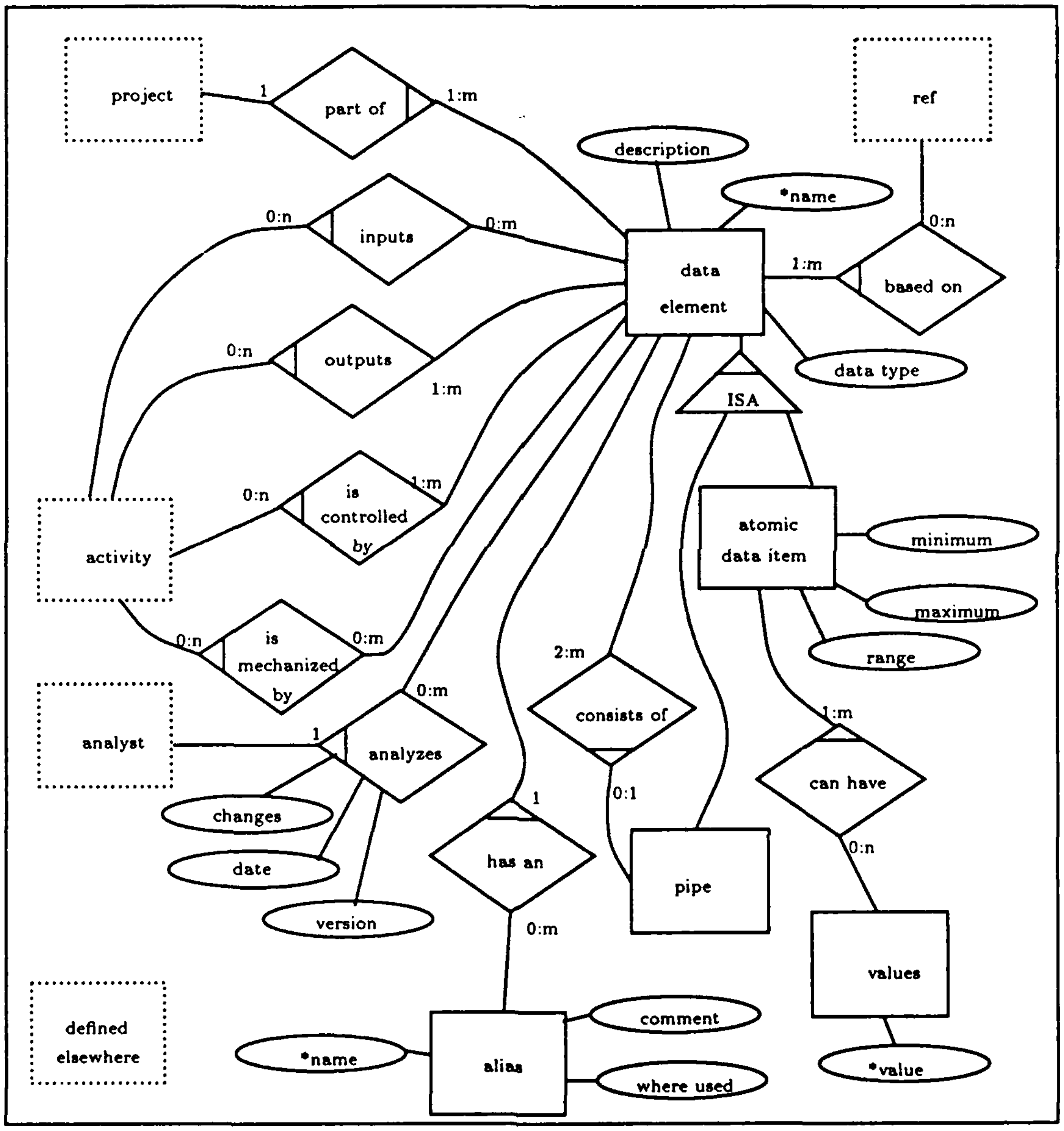

Figure 8: $IDEF_0$ DATA ELEMENT Essential Data Model

Table 1: Description of Components in the Essential Data Model

E–R construct	**description**
activity	This entity represents the $IDEF_0$ (SADT) activities; as noted by the asterisk on the E-R diagram, *node number* is the key attribute, and *name* captures the name of the given activity [7:13–14]. Attribute, *description*, allows the analyst to describe the activity [5:12].
composed of	This relationship expresses the fact that a given parent activity is composed of zero to many (0:m) child activities. It also shows that each activity has one parent activity. The 0:1 notation accounts for the fact that the A-0 activity may not have a parent activity [5:12].
analyst	This entity is used to capture information about the analyst who performed the analysis. The reason for making analyst an entity, rather than an attribute of activity, is so that it might be tied into a personnel database. The entity, analyst, currently has the single attribute, *author*, which identifies the person who performed the analysis [5:12].
analyzes	This relationship expresses the fact that a given analyst analyzes zero to many activities (or data elements). Note that the current model only allows an activity (or data element) to be analyzed by one analyst. Attribute, *version*, is used to record version information; *date* indicates when the analysis was performed; *changes* allows historical data about a given activity (or data element) to be captured [5:12].
project	This entity identifies the project to which each activity (or data element) is assigned. Key attribute, *project id*, is the unique identifier for each project, and attribute, *pname*, indicates the name of the project [5:12].
part of	This relationship indicates that an activity (or data element) is part of exactly one project, whereas a project contains one to many activities.
ref	This entity captures any references associated with an activity (or data element). The key attribute, *reference*, identifies which reference is involved, and attribute, *type*, identifies the type of reference [5:12]. Basically, this entity allows a library of various documents such as DoD standards, user requirements, contractual clauses, etc., to be tied to the given activity (or data element).
based on	This relationship indicates that a given activity (or data element) is based on at least one but perhaps many references, and that a given reference is the basis for from zero to many activities (or data elements).
historical activity	This entity is primarily used as a convenience so that the database does not have to be loaded with analyses which were previously accomplished. The key attribute, *project id*, indicates which project contained the historical activity, and attribute, *node number*, identifies the specific activity within the project.

Table 1 (continued): Description of Components in the Essential Data Model

E–R construct	**description**
calls	This relationship indicates the fact that an activity can call from zero to many previously completed (historical) activities, and that a given historical activity is called by at least one but perhaps many activities [9:33].
inputs	This relationship indicates that an activity can input zero to many data elements. Ross' SA (and the IDEF$_0$ subset) only require activities to have control data elements and output data elements [9:20]. Note that the entity, **data element**, is expanded in the next section.
outputs	This relationship shows that an activity must have at least one but can have many output data elements [9:22].
is controlled by	This relationship shows that an activity must have at least one but can have many control data elements [9:22].
is mechanized by	This relationship indicates that an activity can have zero to many mechanism data elements. Ross' SA (and the IDEF$_0$ subset) only require activities to have control data elements and output data elements [9:20].
data element	This entity represents the IDEF$_0$ data elements; as indicated by the asterisk on the E-R diagram, attribute, *name*, is the key [7:14]. Attribute, *data type*, indicates the type of data (in the Pascal or Ada sense); attribute, *description*, allows the analyst to describe the data element [5:12].
pipe	This entity is a specialized data element, as illustrated via the ISA construct on the E-R diagram. It has no additional attributes, but merely indicates that the data element is actually a pipe containing at least two other data elements [9:20].
consists of	This relationship shows that a pipe consists of at least two data elements, and that a data element can be contained within at most one pipe.
atomic data item	This entity is also a specialized data element for capturing data that have atomic values, i.e., are not pipes. It has three attributes, *minimum* (minimum data value, if applicable), *maximum* (maximum data value, if applicable), and *range* (data value range, if applicable) [5:14]. In the case that none of the attributes are applicable, entity **values**, as described below, probably applies.
values	This entity is used to accommodate atomic data items which have enumerated values, e.g., color can have values red, blue, and green. The entity has a single attribute, *value* [5:14].
can have	This relationship ties the atomic data item entity to its corresponding values entity (if it exists).
alias	This entity captures any aliases that a given data element might have. The key attribute, *name*, is the name of the alias, attribute, *comment*, is used by the analyst to clarify why the alias was needed, and attribute, *where used*, indicates where the alias is used [5:14].
has an	This relationship shows that a data element can have zero to many aliases, and that a given alias corresponds to exactly one data element.

4.2 Drawing Data Model

As we did for the essential data model, the E-R analysis of the drawing data model is done in two parts that complement one another. The first portion of the E-R analysis shows the activity model, with the details about data elements left out. The second portion shows the data element model while leaving out the details about the activities.

Figure 9 illustrates the drawing model associated with IDEF$_0$ activities and Figure 10 illustrates the drawing model associated with IDEF$_0$ data elements. Each of the entities and relationships for both E-R diagrams is explained in Table 2. As appropriate, a reference is given citing why the entity, relationship, or attribute was needed.

5 Summary

The aim of CASE methodologies and tools is to formalize and automate the process of software analysis, design, and implementation so that fewer errors are made and large software problems can be efficiently worked on. The Air Force's graphical analysis language, (IDEF$_0$), combined with a tool like AFIT's SAtool, is an excellent methodology for performing a requirements-phase analysis for any software development project. However, in order to successfully support automated CASE tools, a language must be as unambiguous as possible, with clearly defined semantics for the elements of the language and their allowed interactions. The data model presented in this paper helps to mitigate the ambiguities inherent in the IDEF$_0$ language.

We took a dual modeling approach to formalizing the data model for IDEF$_0$. By dividing the model into an essential and a drawing data model, we are able to separate the analysis data which is "essential" to the requirements analysis and the graphical data which is specific to the display tool. This approach allows the analysis data to be used in an environment that doesn't support or doesn't need the graphical tools of IDEF$_0$ or used with another graphical tool that supports a similar analysis methodology.

Now that a data model for IDEF$_0$ exists, we are investigating its implementation within some type of DBMS. Both relational and nested relational [1] databases are being explored. As this model is difficult to fit into the traditional relational database model, we are interested in seeing if the newer data models will provide a more convenient and efficient database design and implementation.

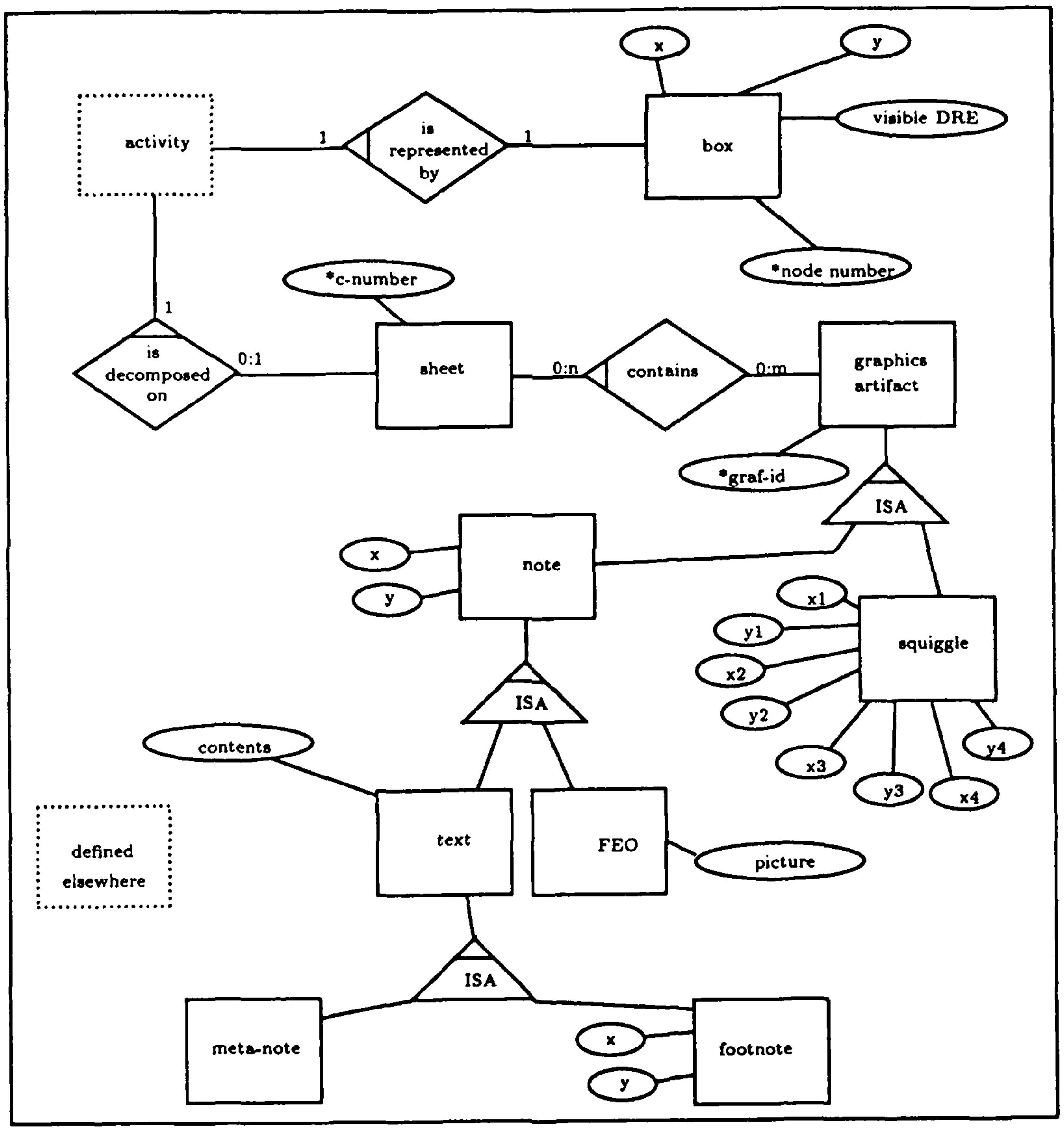

Figure 9: IDEF$_0$ ACTIVITY Drawing Data Model

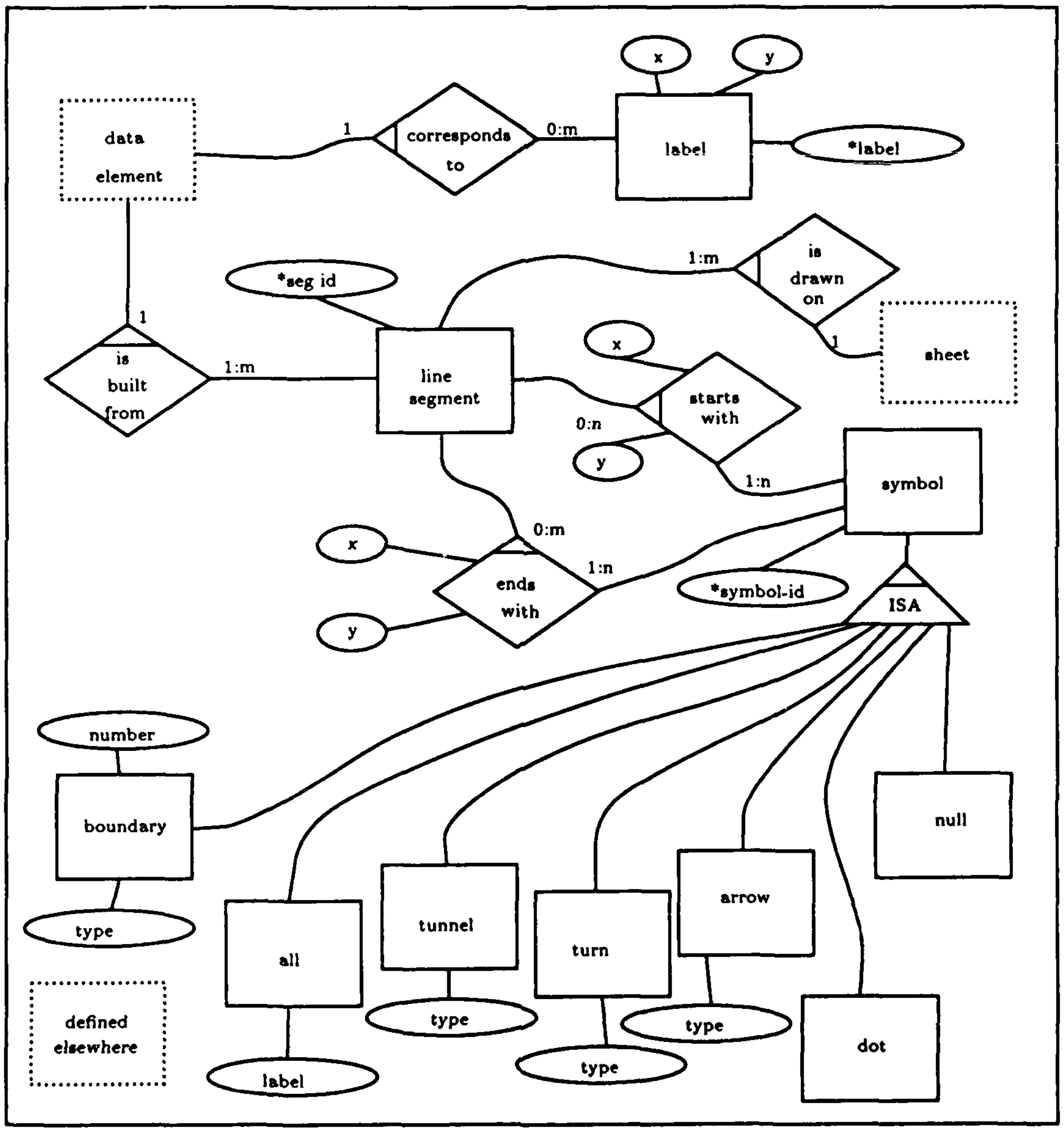

Figure 10: $IDEF_0$ DATA ELEMENT Drawing Data Model

Table 2: Description of Components in the Drawing Data Model

E–R construct	description
box	This entity captures the graphical construct, box, which is used to represent an activity on the IDEF$_0$ diagram. The key attribute, *node number*, ties the box to a specific activity, attributes, x, and y, indicate the location of the upper left hand corner of the box (all boxes are the same size). The attribute, *visible DRE*, corresponds to Ross' detail reference expression. In Ross' words "The omission of a detail reference expression indicates that the box is not further detailed in this model" [9:33]. Accordingly, if the activity being represented by the box is further decomposed then attribute, *visible DRE*, is set to "true."
is represented by	This relationship simply indicates the one to one correspondence between an activity and its graphical representation, box.
activity	This entity is described in the section dealing with the essential data model.
sheet	This entity captures the fact that an activity is decomposed. It has the single attribute, *c-number* [7:17]. Note that *c-number* is used as the DRE symbol on the parent diagram.
is decomposed on	This relationship ties an activity to the sheet upon which it is decomposed, if such a decomposition exists.
graphics artifact	This is a generalized IDEF$_0$ entity which includes notes and squiggles [9:20]. It contains the single key attribute, *graf-id*.
contains	This relationship indicates that a given sheet can contain zero to many graphics artifacts.
note	This entity is used to capture the location of note markers, and it is the generalized entity for both text notes (footnotes and meta-notes) , and FEO [9:20]. There are two attributes, x, and y, which indicate the location of the note marker on the diagram.
squiggle	This entity simply contains the four ordered pairs which denote the location of a squiggle on the sheet [9:20].
text	This entity, which is a member of entity, note, as seen from the ISA construct on the E-R diagram, captures the text for meta-notes and footnotes. Attribute, *contents*, holds the text of the note and the ISA construct indicates the type of note, i.e., footnote or meta-note [9:20].
FEO	This entity, which is a member of entity, note, as seen from the ISA construct on the E-R diagram, captures the drawings associated with the for exposition only (FEO) [9:22].
footnote	This entity is a specialized type of text note as seen from the ISA construct on the E-R diagram. Attributes x, and y, are the location on the drawing where the footnote is placed.
meta-note	This entity is a specialized type of text note as seen from the ISA construct on the E-R diagram.
label	This entity captures the label associated with a data element (the label may be the same as the data element name), as well as the location of the label on the diagram. Attribute, *label*, is the key, and attributes, x, and y, are the location of the first character of the label.

Table 2 (continued): Description of Components in the Drawing Data Model

E–R construct	**description**
corresponds to	This relationship connects a data item to it's label. A data item can have zero to many labels, but a given label can only refer to one data element.
data element	This entity is described in the section dealing with the essential data model.
line segment	This entity captures all the line segments from which the graphical representation of a data item is built. Attribute, *seg id*, is the key.
is built from	This relationship simply indicates that a data element is graphically represented by at least one but perhaps many line segments.
is drawn on	This relationship indicates the sheet on which a particular line segment is drawn. It also indicates that a sheet can have one to many line segments drawn on it.
symbol	This generalized entity is used to capture the type of symbol with which a line segment either starts or ends. Key attribute, *symbol id*, provides a unique identifier for each symbol.
starts with	This relationship connects a line segment to the symbol with which the line segment starts. The two attributes, *x*, and *y*, are the location of the starting symbol. Note that a line segment can start with more than one symbol, e.g., an arrow and a dot.
ends with	This relationship connects a line segment to the symbol with which the line segment ends. The two attributes, *x*, and *y*, are the location of the ending symbol. Note that a line segment can end with more than one symbol, e.g., a boundary arrow.
boundary	This entity indicates that the starting or ending symbol on the line segment corresponds to a boundary. Attribute, *type*, indicates the type of boundary (Input, Control, Output, and Mechanism), and attribute, *number*, is the number of the boundary [7:22].
all	This entity captures the to-all and from-all construct; the single attribute, *label*, captures the to-all/from-all label [9:31-32].
tunnel	This entity denotes that the line segment corresponds to a tunnel arrow; the attribute, *type*, indicates if this is an external arrow that did not appear on the parent diagram (hidden source), or if it is an arrow that touches an activity but does not appear on that activity's decomposition (hidden destination) [7:24].
turn	This entity is used if the line segment starts or ends with a turn. Attribute, *type*, determines the type of turn (right–up, left–up, right–down, left–down, up–right, up–left, down–right, and down–left).
arrow	This entity is used if the line segment starts or ends with an arrow. Attribute *type* determines the type of arrow (right, left, up, and down).
dot	This entity is used in the case of two-way arrows [9:20].
null	This entity is used if the line segment does not have a starting or ending symbol, e.g., the segment simply connects to another segment and therefore does not have a starting symbol.

6 Acknowledgment

K. Austin and N. Smith provided much appreciated input into the initial formulation of the E-R model presented in this paper.

References

[1] ABITEBOUL, S., FISCHER, P. C., AND SCHEK, H.-J., Eds. *Workshop on Nested Relations and Complex Objects* (Darmstadt, West Germany, 1989), vol. 361 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, West Germany.

[2] CHEN, P. P.-S. The entity-relationship model–toward a unified view of data. *ACM Transactions on Database Systems 1*, 1 (1976), 9–36.

[3] CONNALLY, T. D. Common database interface for heterogeneous software engineering tools. Master's thesis, Air Force Institute of Technology, December 1987. AFIT/GCS/ENG/87D–8.

[4] DATE, C. J. *An Introduction to Database Systems*. Addison–Wesley Publishing Company, 1981.

[5] HARTRUM, T. C. *System Development Documentation Guidelines and Standards*, draft 4 ed. Department of Electrical and Computer Engineering, Air Force Institute of Technology, January 2 1989.

[6] JOHNSON, S. E. A graphics editor for structured analysis with a data dictionary. Master's thesis, Air Force Institute of Technology, December 1987. AFIT/GE/ENG/87D–28.

[7] MARCA, D. A., AND MCGOWAN, C. L. *SADT Structured Analysis and Design Technique*. McGraw–Hill Book Company, 1988.

[8] MATERIALS LABORATORY, WRIGHT RESEARCH AND DEVELOPMENT CENTER. *Integrated Computer-Aided Manufacturing (ICAM) Function Modeling Manual (IDEF$_0$)*, report no. UM 110231100 ed. Air Force Systems Command, Wright-Patterson AFB, OH 45433, June 1981. Contract F33615-78-C-5158.

[9] ROSS, D. T. Structured analysis (SA): A language for communicating ideas. *IEEE Transactions on Software Engineering SE-3*, 1 (January 1976), 16–34.

[10] YOURDON, E. *Modern Structured Analysis*. Prentice Hall, Englewood Cliffs, 1989.