Intelligent Tutoring Systems for Systems Engineering Methodologies

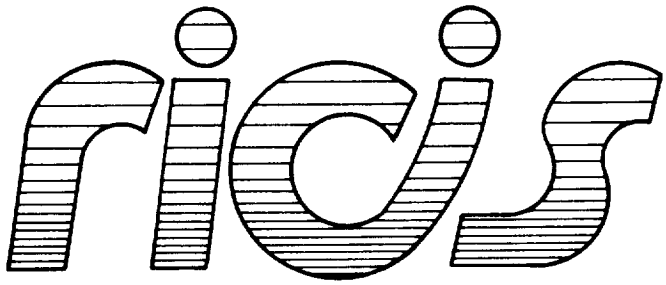
Richard J. Meyer
Joel Toland
Louis Decker

Knowledge Based Systems Laboratory
Texas A&M University

January, 1991

Cooperative Agreement NCC 9-16
Research Activity No. IM.16

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division



Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

Intelligent Tutoring Systems for Systems Engineering Methodologies

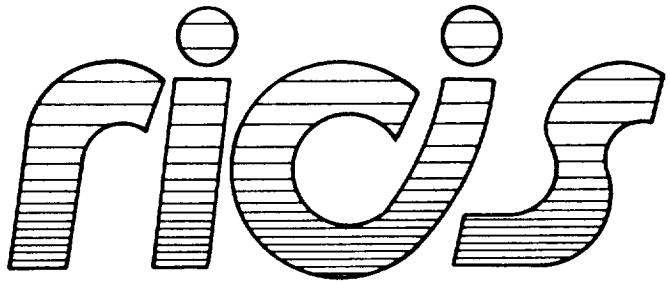
Richard J. Meyer
Joel Toland
Louis Decker

Knowledge Based Systems Laboratory
Texas A&M University

January, 1991

Cooperative Agreement NCC 9-16
Research Activity No. IM.16

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division



Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Dr. Richard J. Meyer, Joel Toland and Louis Decker of the Knowledge Based Systems Laboratory, Industrial Engineering Department, Texas A&M University. Dr. Peter C. Bishop, Director of the Space Business Research Center, UHCL, served as RICIS research coordinator.

Funding has been provided by the NASA Information Systems Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Robert T. Savely, of the Software Technology Branch, Information Technology Division, Information Systems Directorate, NASA/JSC.

Intelligent Tutoring Systems for Systems Engineering Methodologies

Technical Report

January 1991

Prepared by:

Dr. Richard J. Mayer
Mr. Joel Toland
Mr. Louis Decker
Knowledge Based Systems Laboratory
Industrial Engineering Department
Texas A&M University

Table of Contents

1

Introduction

In recent years, demand for use of the IDEF methods has outpaced the supply of both experienced modelers and experienced instructors. A primary use of IDEF models is as a consensus forming communication device among domain experts and between domain experts and system developers. Hence, without consistency not only in practice but also in training and instruction, the benefits of the method cannot be reliably achieved. The objective of this project was to investigate the potential for application of Intelligent Tutoring System (ITS) technology to alleviate these problems. Intelligent tutoring systems (ITS) have received increased research attention in many domains due to the decreasing cost of computer resources. Decreasing hardware costs have enabled economical delivery platforms with sufficient power to integrate artificial intelligence capabilities into ITSs.

Application of the ITS techniques to IDEF training requires application of this technology to a new domain, that of model formulation and model design. Our conclusion after this initial study is that the application of ITS to IDEF tutoring can fulfill the need for consistency in training and overcome the current lack of experienced instructors. However, because of the unstructured nature of the model design process it will be extremely difficult to completely duplicate the capabilities of an expert modeler/instructor. On the other hand, an intelligent IDEF tutor should be capable of teaching a student to model at an initial level of competence. To be successful, an IDEF ITS must develop in the student those minimal analytic, design, and language proficiency cognitive skills required by IDEF modeling.

An intelligent IDEF tutor system could offer several advantages over workbook style (or even classroom style) approaches to education and training in these methods, including:

- personalized interactive style presentation of the information,
- availability of ad hoc query and explanation
- intelligent diagnosis of missed concepts,
- dynamic adaptation of the instructional strategy for effective communication with the student,
- hypertext review capabilities, and

- intelligent evaluation of student's ability.

The goal of this project was to investigate the application of computer assisted instruction (CAI) and intelligent tutoring systems to the development of skills and expertise in the IDEF modeling methods. As part of this investigation we:

- identified the cognitive skills required for $IDEF_0$ and $IDEF_1$ modeling,
- identified the concepts and skills that must be mastered to obtain proficiency in the $IDEF_0$ and $IDEF_1$ methods,
- identified four levels of instruction needed for teaching IDEF based on the recommendations of experienced IDEF instructors,
- conducted a literature review to determine the state of the art in CAI and ITS and corresponding applicable ideas and techniques,
- developed multi-level conceptual dependency (hierarchy) diagrams for $IDEF_0$ and $IDEF_1$ based on the identified IDEF concepts and skills,
- explored integrated expert diagnosis requirements/capabilities for testing students' mastery of a specific concept area,
- explored integrated expert diagnosis requirements/capabilities for the evaluation of a student's general mastery of the material,
- developed an architectural concept for an intelligent IDEF tutoring system that we believe would be applicable to the training of a wide variety of system engineering or analysis methods,
- developed experimental software (on both the PC and the Symbolics Lisp Machine) to test and illustrate/clarify architectural issues and teaching strategies developed.

As additional sources of expertise in the application and training of IDEF methods we consulted Dr. Thomas Cullinane, Mr. Stu Coleman and Major Paul Condit, whose years of experience and knowledge of student modelers have been invaluable in the development of the concepts presented in this report. Further, a paper presented at the May 1990 IDEF User's Group Meeting by Cullinane, McCollum, Duran, and Thornhill provided a synopsis of their training experiences which proved to be very helpful. In their paper, they brought together their observations of the typical characteristics of the $IDEF_0$ student as the student progresses toward becoming an expert modeler.

Four levels of instruction were identified by the research team as promising application areas for an intelligent IDEF tutor. The levels range from a management overview level through readership (model review skills), authorship (model creation), on to an advanced applications level. The management overview level provides a manager with the general picture of the method, but does not provide enough information for the manager to read, create, or review models.

A conceptual dependency network or Concept Dependency Graph (CDG) was developed to represent the prerequisite relations between concepts and skills that must be mastered in the various levels of instruction in IDEF modeling. This structure provides an initial surface representation of an important part of the IDEF training experts' knowledge. We also demonstrated that the developed representation is usable directly as a knowledge base for the dynamic guidance of an IDEF ITS interacting with a student.

We also explored an intelligent diagnostic capability for testing purposes and an intelligent evaluation capability for student modeling. This primarily involved exploring the types of knowledge that should be represented and the types of potential goals for this domain.

Experimental software was developed in order to evaluate issues regarding teaching the modeling practice. Most courses and tutorials tend to focus on teaching the syntactic constructs of a language along with the related semantics. This focus is too limited for modeling however. Methods are designed to keep the syntax and semantics easy to understand. Thus, reading models is usually pretty easy. Modeling proficiency, on the other hand, is usually achieved through experience. The experimental software looks at issues associated with training the modeler in the modeling practice so that a reasonable level of proficiency can be achieved without the modeling experience.

1.1 Research Goals

The general goal of this research is to provide the technology required to build systems that can provide intelligent tutoring in IDEF modeling. This type of capability can extend existing computer support for IDEF modeling beyond model production into training of modeling techniques. Specifically, the research focuses on automated techniques for providing basic proficiency in the syntax and form of the IDEF methods and the strategies involved in the practice of the method.

As mentioned above, teaching the modeling practice is a more difficult problem than teaching syntax. Modeling using the IDEF methods (as well as others) requires the modeler to think in a specific manner. The modeler needs to keep in mind issues like the perspective of the model. It is easy to create countless pages of syntactically correct garbage if one forgets what the model is supposed to show.

Modeling is also difficult because of the general lack of information. Most people have enough trouble describing what they do without even considering trying to describe what their department does. They also usually cannot see the "big picture" of how they fit within the organization. Data modeling is similar. People often believe data is kept when it is not and vice versa. The modeler must be able to extract all of the knowledge a person has and then analyze it relative to other data.

Our analysis of this training domain has led us to conclude that development of the following general cognitive skills in the student is required to produce proficiency in the practice of any of the current IDEF methods (with the exception of $IDEF_4$):

- observation,
- classification,
- abstraction,
- decomposition, and
- language skills.

Observation involves recognizing pertinent elements from documents, interviews, and other source material items. For example, in $IDEF_1$, analysis and observation skills include the ability to distinguish names that refer to objects (physical or conceptual) in the domain from the names that refer to information that is managed about those objects.

Classification involves determination of IDEF concept categories for the discriminated observations. In $IDEF_0$, once activities have been identified, each activity needs to be either classified as part of another identified activity or a new activity must be created which encompasses the lower-level activity (abstraction is described below). While classification is closely coupled with observation it generally proves to be a more difficult skill to master since it entails some of the key model design decision making elements.

Abstraction involves developing an appropriate structure to hold a group of classified elements while decomposition involves selecting the elements from a single structure. Following the $IDEF_0$ example above, it may be the case that an identified activity needs to be grouped with others to form a higher level activity (abstraction) or the activity may need to be broken down into subactivities (decomposition).

Language skills can be broken into performance and competence [Chomsky 65]. Competence is an indication of a person's knowledge of a language (e.g. the IDEF lexicon and grammar). Performance is an indication of a person's ability to communicate using the language (construct models that achieve their application goals). For example, a person may know the words and rules for forming sentences but be unable to communicate a coherent thought. In IDEF modeling, the person may know what activities are supposed to represent but not be able to identify activities of an enterprise. It is important to remember that models are primarily a form of communication. They express concepts which would be more difficult to understand in a linear (textual) format.

For any system to provide effective instruction in these areas, several subgoals must be met including:

1. The system must convey the content of the course in a way that the student can readily understand. This will require the system to be adaptable to the ability and learning preferences of the user. The system will, therefore, need to support various instructional strategies and be adaptable to the student.

2. The system must be capable of motivating the student. This will depend heavily on how the system interacts with the student. Motivational type interaction often can be achieved by providing a variety of positive and negative responses and humorous responses as a form of feedback. Good teachers challenge their students to learn. Thus, a motivational mechanism must be included. A straight forward way to do this would be to provide a "why facility" for each concept. Students frequently want to know why a concept is important enough for them to invest their time to learn it. By showing the student the significance of a concept relative to the whole picture, the desired motivation should be achieved.

3. The system must be capable of recognizing a student's conceptually weak areas, decide what specific concept(s) the student lacks and reteach those concepts. Instead of the system reteaching a lesson using the same instructional strategy to reteach the problem concepts, the system may reteach the lesson using a different instructional strategy.

4. The system should respond differently if the student receives tutoring every day as opposed to once a week. This will be primarily reflected in the review module and in the tutor/student interaction. By using this "temporal trigger" idea, the system will be useful not only in an intense week-long training program but also in an extended month-long training program. The temporal trigger mechanism could be used in combination with other feedback and/or heuristics to alter the system's response to the student.

5. The system should be capable of answering non-predetermined questions. In the IDEF case this must be limited to a particular model for which the tutor system has a preloaded knowledge base of the domain focus of the model.

Approaches to solving these subgoals will be described in detail later in this report.

1.2 Organization of this Report

The report is organized into the following sections and the results of each section are summarized:

Chapter 2 — Background and Current Research Issues

Although background information and a brief look at the state of the art in ITS is presented, this chapter concentrates primarily on the issues related to the student model. The student model is the primary component of an ITS that separates it from a CAI system. Other major issues discussed in this section include the interpretation of test results, dynamic curriculum manipulation, and varying of the instructional strategy.

Chapter 3 — Intelligent Tutoring Systems for Systems Engineering Methodologies

This chapter focuses on ITS issues directly related to systems engineering methodologies. The student's learning goals and known diagnosed problem areas provide a context in which to customize the training. Pedagogical levels are defined and Concept Dependency Graphs (CDGs) are explored. The CDG is a key component used by the IDEF tutor to represent domain structure. Finally, a way is presented to actually reuse lessons represented by the CDG from course to course. We believe this will be a key factor is the reduction of ITS development costs.

Chapter 4 — IDEF Tutor Architecture and Components

The eleven components of the IDEF ITS architecture are described - from the user interface and student model modules to the lesson files and model builder environment. Descriptions are conceptual in nature with implementation details reserved for later chapters.

Chapter 5 — Developing Cognitive Skills for IDEF Modeling

To get around the problem of tutoring an unstructured domain, special effort is required to develop the cognitive skills for IDEF modeling. This section focuses on two approaches to that problem. The purpose of a library of models is also discussed along with some further comments.

Chapter 6 — Experimental Software

Experimental software was developed on the Symbolics Lisp Machine to experiment with various ways to allow the student to develop the cognitive skills discussed previously. This section describes that software, the results, and our evaluation of the effort.

Chapter 7 — PC Based Prototype

Building upon the research efforts of this project, a PC-based prototype is being developed to test out the architecture, test the effectiveness of the cognitive skills approach, and provide a mechanism for further experimentation. The prototype is

being developed as part of the requirements toward a M.S. in Industrial Engineering by Joel Toland at Texas A&M University.

1.2.1 Conclusions

This section summarizes our conclusions and results. It also addresses future directions and potential work areas for further research and development. For example, the use of the Delphi technique for knowledge acquisition of the model libraries is discussed.

2

Background and Current Research Issues

Computers have been used in education since the early 1960s. Claims of their role in improving education have been heard for almost 30 years. Until recently, CAI systems were hampered by hardware limitations and a behavior theory approach which "poorly matched the cognitive goals of education" [Mandl & Lesgold 88]. Early tutorial programs were noted for their rigid structure, their prespecified lesson content, and their inability to adapt to the user's needs.

Training students in the IDEF methods is complex at best without taking into account their backgrounds. Unfortunately, students are not blank sheets of paper waiting to be the "right" way. Most students will have modeling backgrounds using one particular method. It is common for people to latch on to one method and try to use it for whatever they need. When that student is introduced to the new method, the student will compare it to the previously learned method. Subtle differences in terminology and syntax may cause serious misconceptions on the part of the student.

Due to this problem and others an IDEF tutor cannot follow a rigid plan like early CAI systems. The system must adapt to the student and watch for tendencies on the part of the student to make misassumptions based on previous knowledge. The system must also be able to skip areas the student already understands from training in other methods and backtrack to correct earlier misconceptions.

As early as the mid-60's, Uhr developed systems to generate problems in arithmetic [Uhr 69]. If the needs of the user matched what the program had to offer, the program was successful, otherwise it was not. Traditionally, computer-based learning systems have ranged from systems in which the system maintains control (e.g. Plato) to systems in which the student maintains control (e.g. Logo) [Soloway & VanLehn 87]. The first type of system restricts the student's ability to explore the concepts while the latter allows the student to explore at will but lacks structure. Both the computer dominant and student dominant programs are too extreme to be suitable for teaching. Intelligent tutoring systems (ITSs) represent a mixture of the two extremes in which the tutor and the student share control of when, how, and what is being taught [Soloway & VanLehn 87]. The result more closely models the interaction between a human teacher and student than either of the aforementioned extremes.

2.1 State of the Art in ITS

The development of ITS systems has been constrained to domains which are highly structured. Such domains are logical choices for ITS development because their natural organizational structure maps well into "rules" for the system to operate under. The more sophisticated systems use a combination of an embedded expert system, simulation, and natural language interface.

The majority of intelligent tutor systems have been developed for teaching skills in structured domains such as:

- mathematics (addition, subtraction, geometry, algebra),
- programming languages (LISP, Pascal, design issues),
- electronic troubleshooting [Brown, Burton, & DeKleer 82],
- medical diagnosis [Clancey 81], and
- chemistry [Lee 88]

In these domains, an ITS generally includes an internal representation of the domain that it manipulates to solve new problems. This capability allows the ITS to verify the correctness of a student's answer to a nonpredetermined problem. In a domain such as programming languages, the syntax and to some extent the semantics are known to the ITS. Although such systems can locate syntax errors much like a compiler would, they really cannot verify the semantic usage of the language. The best such systems currently do is to watch for specific kinds of errors and make suggestions.

The merging of simulation and expert systems techniques has demonstrated tremendous potential. A drawback of expert systems is their inability to make decisions based on time. In a business situation, for example, a dollar today does not necessarily have the same value as it did yesterday. Simulation, on the other hand, is very good at modeling situations where time is constantly changing but cannot interpret the results it produces. A combination of the two techniques is a system that can simulate the future and make longer term decisions instead of decisions based only on the current knowledge base. This capability should prove valuable to intelligent tutoring systems.

In conclusion, the state of the art in intelligent tutoring systems is far from sufficient to address training students in modeling skills. Current systems do not have the flexibility, modularity, sophisticated knowledge bases, and student models necessary to cover such difficult issues as knowledge acquisition. It should be noted that human instructors have trouble teaching good modeling practice. Trying to create an ITS which does it well will be just that much more difficult.

2.2 Modeling the Student

Defining the "Intelligent" part of ITSs is as difficult as defining "artificial intelligence" — there is no commonly accepted definition. VanLehn notes that many ITSs "infer a model of the student's current understanding of the subject matter and use this individualized model to adapt the instruction to the student's needs" [VanLehn 86]. We will use this statement to characterize ITSs.

The student model is one of the major differences between a CAI system and an ITS. The ITS attempts to maintain a representation of what it perceives to be the student's current level of knowledge/understanding. This representation is updated as more information is gathered about the student and used to dynamically make decisions concerning interaction with the student. CAI does not typically maintain a model of the student. Information is typically saved in a file to reflect the student's progress but only general type information is maintained that affects the interaction with the student. With this capability, an ITS, for example, can be attuned to the way the student tried to solve a problem while a CAI system is generally only concerned with the final answer. An ITS might try to match the student against internal models of different student types, so it can tailor its interaction with the student in an appropriate manner. A CAI system generally treats all students the same.

As mentioned earlier, students learning an IDEF method will probably have some experience with different methods. It will be important that the tutor system be able to watch for certain tendencies on the part of the student. For instance, if the student had been a data modeler for some time and then wished to learn IDEF1 to do information modeling, the system would watch for tendencies on the part of the student to try to do database design instead of just modeling what information is kept by the enterprise.

There is some confusion in the field regarding the meaning of "student model". At times it is used to refer to a trace of the student's responses, while at other times it refers to the "incorrect" answers and the trace. In this research it has a completely different meaning. The student model is a model of the student's behavior under various levels of misunderstanding.

Closely related to the student model is the diagnostic component of the ITS. The diagnostic component must infer the student model based on the feedback from the student and interaction with the student [VanLehn 86]. Such inferences are difficult to draw when teaching method practice since there is really never one "correct" way. Each modeler develops his or her own techniques which fit with the way the individual thinks. It may take many steps into building a model before the system can really say that a mistake has been made. At such a point, the system would have to backtrack to find where the student started to go astray. A model of the student's behavior will necessarily be quite complex.

2.2.1 Potential Application of Student Models for IDEF ITSs

A student model can be used to improve the performance of an ITS in several ways [VanLehn 86]:

- advancement — the student model represents the level of mastery for concepts; this is useful in situations where the student must manipulate one or more skills or concepts at one time,
- problem generation — ITSs that generate problems consult the student model to match the problem level with the student's current capabilities,
- adapting explanations — an ITS should not use concepts in explanations that the student has not mastered yet; by consulting the student model, the ITS can verify that its explanations are appropriate for the student's current capabilities.

Since teaching IDEF modeling is so complex, each of these strategies would need to be taken advantage of. Such dynamic systems would require a strong underlying architecture if they are to be successful. The primary functionality for advancing the student through the concept hierarchy, generating problems, and adapting explanations will need to be built into the base system so that lessons can be generated in a reasonable amount of time. This functionality will require state of the art knowledge based systems along with advances in cognitive modeling. Later in this report we will identify the key components of such an architecture.

2.2.2 Diagnostic Techniques for Student Models

Since the diagnostic component must infer the student model based on the feedback from the student and interaction with the student, the diagnostic component of an ITS should be tightly integrated with the student model. Much of the information in the student model will be inferred data from the diagnostic component. VanLehn has noted that nine diagnostic techniques have appeared in the ITS literature [VanLehn 86]:

- model training [Anderson, Boyle, & Yost 85],
- path finding,
- condition induction [Langley & Ohlsson 84],
- plan recognition,
- issue tracing,
- expert systems,
- decision trees,
- generate and test, and

- interactive diagnosis.

Our development team concluded that only a few of the aforementioned diagnostic techniques are directly applicable to an intelligent IDEF tutor. The selected techniques will now be examined.

2.2.2.1 Plan Recognition

Plan recognition is the identification of causality rationale or precedence rationale for the history of a set of actions/responses taken by the student. Plan recognition serves as "a front end to model tracing" when used for diagnosis and "requires that the knowledge in the student model be procedural and hierarchical" with most of the "physical, observable states in the student's problem solving" accessible [VanLehn 86]. VanLehn also noted that plan recognition is similar to parsing a string with a context-free grammar. Genesereth also explored ways of discovering a student's problem solving "plan" and how that plan could be used to generate appropriate remedial instruction [Genesereth 82].

Although it will be difficult to implement, recognition of a student's plan for specific IDEF modeling situations may be possible. Plan recognition is critical in teaching modeling practice since the system cannot let the student get so far down the wrong road that backtracking becomes difficult. The system would not necessarily have to understand each individual situation, but instead would watch for tell-tale signs of a misguided plan. An analogy can be made to grammar checking programs. A grammar checker can often find problem areas because it is looking for specific situations for which it has an applicable heuristic or rule. It can find many errors even though it does not "understand" the semantics of the sentence. By constraining the student to modeling specific known situations, the task may be simplified although it is still difficult.

2.2.2.2 Expert Systems

Using an expert system as the "domain expert" in an ITS system initially seemed like a logical approach to ITS researchers. Some success has been achieved in domains characterized by rigid rule structures. The approach is very difficult to apply in general. The purpose of this section is to relate ideas which have been spawned from research into expert system application to ITSs. It is unlikely that there will be an expert system capable of modeling in any particular method, but the ideas are still worth while.

A tutorial session at AAAI-87 [Soloway & VanLehn 87] dealt with the question of using an already developed expert system (MYCIN) as part of an ITS. The strategy was to compare the student doctor's responses with MYCIN's responses. MYCIN was designed to solve problems and therefore its expertise was compiled into the system. It was discovered that, for teaching, a "decompiled" knowledge base was needed to represent the knowledge explicitly. Compiled knowledge is knowledge

which has become so specialized to a specific use as to have lost transparency and generality [Wenger 87]. Decompiling the knowledge base, in this case, is defined as explicitly separating the diagnostic strategies from the disease knowledge. Based on this information, a good approach for an IDEF tutor seems to be separation of the diagnostic heuristics from the domain representation.

NEOMYCIN was the second expert system examined since it contained a decompiled rule base. NEOMYCIN separated out diagnostic strategies and disease knowledge. The conclusion was that it is very difficult to turn an expert system into a tutoring system because of two reasons:

- the rule base for problem solving is inadequate for teaching
- need to explicitly represent what needs to be taught

Although an expert system does not currently exist for IDEF, an expert system might still be used in the IDEF tutor. A small expert system could be used to make intelligent decisions concerning when to change instructional strategies.

Lee, from the Chinese University of Hong Kong, has proposed an expert system construction methodology for developing intelligent courseware [Lee 88]. His approach uses an expert system shell to provide a flexible environment for CAI development. Such a system only requires the development of the knowledge base for the particular subject domain. Lee's system is written in Prolog and consists of a knowledge editor, a tutoring module, user interface, and a question-answering module. The advantages of the system include its ability to answer non-predetermined questions with non-predetermined answers and to provide problem-solving and explanation-giving capabilities. Such a system is appropriate for a domain characterized by a rigid rule structure like chemistry or math. It falls short, however, when applied to areas like IDEF function and information modeling which are not constrained by physical or mathematical laws. Lee provides no discussion or evidence to indicate the effectiveness of the system with students.

Lee's ideas can be used to build a system for course preparation. The Concept Dependency Graphs for the various methods are going to have similarities which could be identified by an expert system. The expert system may be able to point out areas which are not covered in a course which are generally covered in courses for other methods. The expert system may also be able to critique the flow through the concept hierarchy, such as pointing out places where a term may not have been covered previous to its use in an explanation. There are surely other parts of the course preparation which could be aided by an expert system.

2.2.2.3 Decision Trees

In 1978, Brown and Burton proposed the "Buggy model" in which errors by the students were seen as symptoms of a "bug" [Brown & Burton 78]. Bugs were

considered to be discrete modifications to the correct skills which duplicate the behavior of the students [Sleeman & Brown 82]. Genesereth, Brown, and Burton focused on how the student went about solving a problem so that the misconception could be corrected. This approach is possible but difficult to implement for IDEF in a rigorous fashion. If a significant amount of expert level heuristics may be obtained, a variation of this approach could prove useful. For an IDEF tutor, this approach will depend heavily on the existence of a Concept Dependency Graph or CDG (see Section 3.6) for guidance. An attempt to pinpoint misconceptions may be made by utilizing the CDG's dependency relationships to decompose the suspected missed concept. Testing can then proceed recursively using the subconcepts to isolate the misconception. Correctly identifying misconceptions is very difficult but, if successful, would pave the way for any attempt to repair the problem.

In conclusion, plan recognition and decision trees can be used to facilitate the student model diagnostic capabilities of the ITS. Expert system strategies could be used to aid in creating and diagnosing the CDG and possibly to diagnose the student model.

2.2.3 The IDEF ITS Student Model

Maintaining information about each student is essential for the system to "know" the student's learning goals, modeling experience, and progress on previous lessons. Just as a human instructor builds an implicit mental picture of a student's ability and knowledge, a computer based tutor should build a model of the student This will take the form of a student model maintained in a database. Figure 2.1 illustrates the types of information that an IDEF ITS student model should maintain.

```
First Name: John
Last Name: Smith
Learning Goals: reader, author, reviewer, advanced applications
Occupation: manager
Background: IDEF0, DFD
Progress:
  [11] ((3/5,4/4,7/8) (60%,100%,87.5%) (82%)),
       finished 2/12/90 09:30
  [12] ((6/10, 9/9) (60%,100%) (79%)),
       finished 2/12/90 10:38
Skipped: 1,2,3,4,5,6,7,8,9,10;
```

Figure 2.1 Example student model file

The student model file maintains information to identify the student by name, learning goals, occupation, and background. It must also maintain information on the student's progress. If the tutor knows that the student has a specific type of modeling background, then analogies and examples which build upon the student's previous knowledge can be used to facilitate the learning process.

In Figure 2.1, a hypothetical student's progress is recorded. For the first concept (denoted by "11" in this case), the student scored 3/5 (60%), 4/4 (100%), and 7/8 (87.5%) on the subtests, finishing at 09:30 on February 12, 1990. Overall, the student scored 82% on that test sequence. Similar progress is recorded for the second concept (denoted by "12").

Acquisition of the initial model information is accomplished by questioning the user directly. Alternatively, a supervisor of the tutor system could setup each student's model based on knowledge of the student's abilities, background, and learning goals. Other information, such as the student's progress, is gathered as the student interacts with the tutor. Additionally, the information from the user models could provide feedback about the system/student interaction as well as data for research purposes.

2.2.4 Student Model Example

The Intelligent Maintenance Training System (IMTS) is an example of one system that attempts to maintain information about the student and adapt based on the acquired information. A student model is "maintained to aid in problem selection and other student-specific decisions about instruction"[Towne & Munro 87]. Three kinds of data are used:

1. moderately detailed representation of the student's conceptual model of the equipment being taught,

2. simple global measures of student competence and learning preference (preauthored for each student by instructor), and a

3. detailed breakdown of tests performed for the current problem.

The first type of information in the IMTS student model depends on a normative approach to representing student knowledge and skills. This approach relies on a normative model of an expert's understanding of the domain. In this case, the domain is equipment to be maintained. The model is in the form of a tree in which each node represents specific knowledge about some aspect of the equipment.

The second type of information in the IMTS student model consists of global measures of competence and learning preferences. These measures are setup beforehand for each student and actual performance on practice problems is supposed to overcome any instructor bias to establish the appropriate values.

According to Towne and Munro, "a model for an individual student consists primarily of an updated set of mastery levels for the knowledge/skill elements in the structure" whose "values are changed to reflect troubleshooting problem performance as the student progresses through the problem curriculum" [Towne & Munro 87].

The student model of the IDEF ITS will keep the same types of information as the IMTS. Since, the IDEF tutor deals with a more complex subject, the models will consequently need to be more sophisticated. It is important to recognize that the IDEF tutor builds upon the experiences of experts in the industry.

2.2.4.1 Acquisition and Understanding of Student Models

Human instructors know and/or assume information about their students when they prepare to teach a course. An instructor preparing to teach a course, for example, knows the required prerequisites for the course. In college, the student's major is usually known. In high school, the student's "track" and grade level is either known or assumed. Tracking refers to the student's overall curriculum orientation such as a college preparatory track or a vocational track. The instructor, therefore, indirectly knows or assumes the background and learning goals of the students.

Student's learning goals may vary considerably. A student may be required to take a course that the student considers useless. Another student may see the same course as preparation for a more advanced course. To another student, the course may be an advanced course. Human tutors are able to dynamically adapt to the learning goals to enhance the learning process. For example, a professor can teach a course many different ways using the same textbook. The professor might teach the undergraduate version of a course with an applied focus while teaching the same course to a graduate class with a theoretical focus. Current systems generally do not adapt to the learning goals of the student. For an ITS system to be effective, it should be capable of adapting its curriculum to the learning goals of the student. The system will have to operate on the assumption that the student wants to learn but at varying levels of detail.

Throughout a course, the instructor creates a model of each student in several ways. The results of examinations and quizzes are one measurable way. Verbal feedback is another way. The model created by the instructor is used to determine the student's grade and to aid the instructor in understanding what areas are still weak. All in all, it is the indicator of what the instructor feels the student knows. For an ITS system to "understand" and evaluate what the student knows, the same type of knowledge must be captured.

How will a student model be acquired and used by an ITS system? Actually, the initial information acquisition should not be difficult. It could be entered through menus by either the student or a system supervisor. The student's learning goals and background information will, by necessity, be selected from a series of general

options. The learning goals, for example, should match the pedagogical levels available. Additional information should be collected to enhance the tutor/student interaction. The model will be updated as the student is tested to reflect the student's level of understanding.

What information is essential to a student model? First, a means of uniquely identifying the student is needed. Second, the student's learning goals are needed. Such goals should be general instead of specifically concerning the domain. For example, does the student want beginning, intermediate, or advanced concepts? Next, information about the student's background should be used, though how this information should be used still requires considerable discussion. The student's background will aid the system in traversing through the CDG.

Finally, feedback from the system should be used by the student model. This should extend far beyond results of testing to include suspected problem areas and information about how the student learns. As students use the tutor, a database will be created to keep such information as average time to complete each lesson, number of times each lesson is presented, and the effectiveness of the various instructional strategies for each lesson. The more the student uses the system, the more data the system will have to use as a basis for modifying its behavior to increase its efficiency as a tutoring system. After enough students use a tutor system, patterns may emerge as to which instructional strategies are more effective for each lesson. Effective use of such feedback has the potential to help create a self optimizing tutoring system. It may also be used to pinpoint poorly designed lessons.

Interpreting and utilizing the student model will be difficult. A number of questions arise as a result such as:

- *If the student does not appear to understand a concept, should the student be allowed to continue if the student wishes to override the system and continue on to the next concept?* For instance, if the student wanted an introduction to modeling practice but will not necessarily be doing any modeling, less than passing scores may be sufficient in those lessons.
- *How should this information be used if the student performs poorly on more advanced topics?* Lessons should be kept as independent as possible, but if there is a strong dependency between concepts, then the student must understand all related concepts to move on. In the example above, if the student pays little attention to modeling practice, then the student will be limited in advancement in areas which require those abilites.
- *Should the final evaluation of the student note the trends and possible weakness in specific concepts?* In other words, when the student finishes should there be more than just a grade? The student should be characterized as best as possible without drawing rash assumptions about the student's abilities. Some people may be better than others at syntactic understanding,

but may not be as good at modeling. It is important for the system to note these abilities.

Flexibility is important for any system. When the system is initialized for use and set up for specific students, the system's human supervisor should provide information to tailor the behavior of the system for those students. For example, should the system evaluate the student's knowledge and automatically place that student at an appropriate place in the course? Does the student have any experience or knowledge that may be exploited when teaching a new subject through analogous examples?

2.3 Curriculum and Content

> *Systems developed thus far have failed to adequately represent the structure of the curriculum being taught, concentrating instead on trying to represent all the knowledge to be taught.* [Lesgold 87]

The IDEF ITS seeks to address this problem. Past systems are somewhat similar to teachers who are experts in their field but do not know how to motivate and teach their students. Through the CDG and other strategies the IDEF ITS provides structure for the curriculum.

More learning theory needs to be built into ITS systems. Gagne made several points that are relevant. The first was that learning is more than simply the sum of its parts. Specifically the lowest-level subgoals in a goal hierarchy do not, as a group, contain all the knowledge implied by the highest level goal. The combining of the subgoals results in new knowledge. Gagne calls this new knowledge the "glue" that ties together pieces of knowledge [Gagne 62]. Unfortunately, this "glue" is often difficult to identify and is the difference between why one teacher is effective at explaining concepts while another is not. For an intelligent IDEF tutor system, the pieces of knowledge will be represented by a concept hierarchy. The "glue" which ties these concepts together will be in two forms:

- carefully selected case studies from expert modelers to gain as much hands on modeling experience as possible, and
- development of the cognitive skills necessary for IDEF modeling.

Developing an ITS system based on traditional models of teaching may actually be restrictive in several ways. The manner in which students are taught in schools is determined more by management constraints of the school and classroom than instructional theory constraints. For instance, students in the traditional classroom setting are tested and graded as a group, whereas most likely only a single student will use an ITS at a time. Thus ITS systems may or may not be forced to live under the same set of constraints as traditional models. Testing, for example, may be dynamic in nature. During testing, the system can attempt to explore suspected problem areas in a very dynamic, individualized way that is generally not possible

in a large classroom setting. The system can also provide feedback to the student and or teacher immediately. Even the way a subject is taught may be different. Computer simulations can dynamically make large things small, small things large, slow things faster, and fast things slower. A student may interact with such simulations in a completely different way than is possible in a traditional classroom setting. Care must be taken not to fall into the trap of designing the IDEF tutor system based on an unnecessarily restrictive model of teaching.

One of the primary explorations for the experimental software was to determine what type of interaction is successful in a computer to student relationship. It may be that practice can take a larger portion of the responsibility for instruction since the student is receiving constant attention from the instructor. In a large classroom, practice is of limited value since the instructor cannot provide immediate feedback to the student. Simulations of interview sessions as well as model construction will be important to the development of the modeler.

2.4 System Supervisor

The role of the system supervisor (the person who sets up the system) of the ITS is important. An initial student model is needed so that the ITS can identify each student. A system supervisor can not only customize the system but also verify that the student models are initialized correctly. Conversely, for single user systems, the student would serve a dual role as the student and system supervisor.

The student model accesses parameters which are given values at set up time to customize the behavior of the system. Typical parameters are those that affect how the system interacts with the student. An example would be the expected student/tutor interaction if the student scores poorly on a test but does not want to attempt to learn the concept a second time. Another example would be setting thresholds for test scores above which the student must perform on each of the subsection tests and/or on the average of the subsection tests.

2.5 Interpretation of Test Results

The ability of the system to dynamically alter the curriculum necessitates the need for appropriately varying interpretation of test results. For example, advanced levels may require different threshold values to be appropriately established by the supervisor. Although four levels of student expertise have been identified for $IDEF_0$ and $IDEF_1$ modeling, the number of levels may vary from course to course. For the author level, higher threshold values could be set since the student must be able to create models not just read them. In other words, the students could be required to score 90% on testing of the basic concepts while requiring 80% on the most advanced concepts. Again, the idea is to make the system as flexible as possible. At this point,

the issue is not how threshold values should be set, but rather the fact that such functionality should be incorporated into the system.

A "for your information only" type level may or may not require any testing. The system could be set up so that at a certain level, it does not test the student. This behavior would be set by the system supervisor. Such a lack of testing would allow a student to acquire knowledge "for their information only". Material covered in this manner is generally not tested when taught by human instructors. This level should be equivalent in content to a seminar, an introductory overview, or a briefing. The student model will therefore only reflect the fact that the student has completed (in part or whole) the level. This could be especially useful, for example, to brief a manager. The management overview level embodies this concept.

At this point, thresholds for test interpretation need to be addressed. Although human instructors are able to assign grades (e.g., A,B,C,D, and F), requiring a computer to accurately interpret human test results with such granularity would be very ambitious if even possible. A pass/fail grading system based on threshold values is a more reasonable alternative. Threshold values may be set by the system supervisor for each test and subtest. Alternatively, default values may be used. Two thresholds may be set: a subtest minimum threshold and a test threshold. The test threshold is equal to the sum of the subtests weighted appropriately. This two level threshold system is designed to catch specific areas where the student does not understand a concept and test the student's ability to comprehend the concepts when they are integrated. The threshold values would be used by the student model module to decide whether to test a student further to isolate a potential problem or to allow the student to proceed to the next lesson.

2.6 Dynamic Curriculum Manipulation and Varying Instructional Strategy

Although an ITS cannot dynamically generate an entire course curriculum, it can do something very useful given the dependency relationships between the concepts. By knowing the student's learning goals, the system can vary the depth of knowledge to match. If a student wants introductory seminar level knowledge, a considerable amount of detail should be suppressed. Conversely, for the most advanced level of instruction, no detail would be suppressed. If the ITS system knows about the student's background, it can alter the types of examples. If it can test the student and diagnose the student's problem areas and evaluate the student's current level of knowledge, however, then it should alter the curriculum in a more sophisticated way.

For example, assume that a student uses an intelligent $IDEF_1$ system. If the student has been working with $IDEF_0$ for a year, the student should already know concepts such as the roles of the modeling group members and how IDEF kits are put together. There is no reason to reteach the information to that particular student unless the

student shows some knowledge gap in that area. Testing the student to discover a starting point could reveal problem areas. Students often believe that they understand a concept, but when tested, find that more instruction is necessary.This dynamic curriculum adjustment provides a dynamic way for the system to meet the student's learning needs in an "intelligent" fashion.

The "auto-placement" of a student in the course proceeds in one of several ways. The tutor may test from the goal concept backwards toward the fundamental concepts or vice versa. Intuitively, testing from the fundamental concepts toward the goal concepts seems more practical. Questions that must be answered include:

- how many questions should the student be asked to estimate the student's current level of knowledge?
- how should the questions be choosen?
- could the questions for the auto-placement be auto-generated based on the knowledge in the CDG that relates the testing and lessons?
- based on the testing results how should the student then be placed?

The CDG guides the ITS system in several ways. For instruction, the CDG is used to determine which concept to teach next. It is systematically traversed from a base level to the final concept. For testing, given an intermediate target concept in the CDG, the CDG is traversed backward testing the subconcepts that must be learned to understand the target concept. If necessary, testing continues recursively level by level until a specific subconcept the student does not understand is isolated or the system is unable to isolate the problem area.

Manipulation of the CDG alone does not change the instructional strategy used to teach the lesson. At the lowest level, the lessons must, by necessity, be hardcoded to some degree. By providing multiple copies of the same lesson each based on a different instructional strategy, multiple instructional strategies are supported. Although this requires more work to develop the lessons for the system, this strategy avoids the problem of locking the student into one learning model, and consequently increases the flexibility of the system.

A case can also be made for systems that allow the student to choose what lessons to learn. That mode of usage should also be provided a facilitate browsing and manual use of the system.

Hopefully, by this point the important concepts of ITSs have been addressed as well as the relationships between those concepts and the current research. It should be clear that previous ITS and CAI efforts have not raised the level of sophistication of the technology enough to build an ITS for IDEF. In the remaining sections we will focus more strongly on goals for ITSs for systems engineering methodologies,

describe the IDEF ITS architecture, describe the experimental software and PC prototype and then draw conclusions on the research effort.

3

Intelligent Tutoring Systems for Systems Analysis Methodologies

3.1 Problems With ITS Applied to IDEF Systems Engineering Methods

3.1.1 Goals

The obvious question that arises when discussing the development of a system, such as an ITS for IDEF, is "why is it so hard?" It was mentioned earlier that the domains chosen for ITS systems are generally very structured. Domains such as mathematics, programming languages, and chemistry have very rigid rules associated with them. Teaching a systems design methodology is considerably more difficult due to the qualitative nature of the methodology. Merely creating a slide show to present the material will be no more informative, though possibly more effective, than handing someone a manual with the instructions to read it and figure it out. Several requirements for an effective ITS system have been identified based on previously constructed systems, research, and the experience base of the development team. An intelligent IDEF tutoring system should be capable of:

- acquisition and maintenance of the student's learning goals and relevant background
- assessment of the student's present knowledge
- determination of how the student's present knowledge maps into the material to be taught
- adjusting the strategy for interacting with the student
- motivating the student
- testing the student and interpreting the results
- demonstrating correct vs incorrect models by examples.

The present strategy for achieving these goals involves modeling the student, diagnosing the problem areas, relating curriculum and content, and developing the required cognitive skills. Applicable cognitive skills include observation, classification, abstraction, decomposition, and language skills.

3.1.2 Problems with Computer Aided Modeling

Teaching a student to model in an IDEF method involves not only teaching the syntax and semantics of the method, but also hands on experience and learning from mistakes. The question "why is it so hard?" takes on a new level of meaning when one tries to evaluate a student's model. It is difficult to automate the evaluation of a model because it is difficult to evaluate creativity and insight.

Expert modelers/instructors stress the importance of hands on modeling experience to learn, yet this problem poses the largest stumbling block for an IDEF tutor. Although there are tools available for aiding the model development process for IDEF, these tools cannot evaluate the quality of a model. Their only function is to facilitate model entry, development, and output much like a computer aided design tool facilitates the drafting process.

Strategies are needed which will provide each student with feedback concerning individual modeling efforts without going beyond the limits of today's hardware and expert systems technology. One such strategy, which will be discussed again later, is to provide each student with previously developed models which the students can use for comparison. The students can then identify the good and bad points of their models.

3.2 Rules for IDEF$_0$

Although IDEF$_0$ modeling may be considered as much an art as a science, the methodology has rules that must be followed. IDEF$_0$ rules include [Mayer 90]:

- detail exposition control at each level (3-6 box rule),
- bounded context (no omissions or additional out-of-scope detail),
- diagram interface connectivity (node numbers, box numbers, c-numbers, and detail references,
- data structure connectivity (ICOM codes and use of parentheses),
- uniqueness of labels and titles (no multiple names),
- syntax rules for graphics (boxes and arrows),
- data arrow branch constraint (labels for constraining data flow on branches),
- input vs. control separation (rule for determining role of data),
- data arrow label requirements (minimum labeling rules),
- minimum control of function (all functions require at least one control), and
- purpose and viewpoint (all models have a purpose and viewpoint statement).

Listing some of the rules for IDEF0 highlights the problem at hand — the difficulty of automating the task of verifying a student's models and evaluating their worth. Although checking most of the listed IDEF0 rules can be automated, syntax checking alone is not worth much without the ability to verify the semantics. This point will significantly affect the approach we use to teach IDEF modeling.

3.3 Understanding the IDEF0 User: Trainee to Expert

This section is grounded on the work done by Cullinane et al. [Cullinane, McCollum, Duran, and Thornhill 89]. Their work is the most recent compilation of expert modelers/instructors describing the typical IDEF0 student in terms of specific tendencies and abilities at various levels of proficiency. It is also important to note that Cullinane, McCollum, Duran, and Thornhill do not represent the interests of a single group or organization; they are from different organizations and different parts of the country. Their paper draws from their observations made over many years IDEF0 instruction. We have found these observations useful during development of the architectural components related to student modeling and testing.

3.3.1 What Makes A Good IDEF0 Modeler?

Good modelers are generally characterized by good interpersonal skills, communication skills, and an ability to abstract. Effective written and oral communication skill are essential. An IDEF analyst must often communicate with a wide variety of people to collect information and be able to express the relevant information in written form (summary briefings) and in a model. The ability to abstract is very important for IDEF modeling. Often a modeler must sift through a tremendous amount of data and then be able to abstract the relevant information to create a useful model.

Cullinane et al. also recommend that "The class should be taught by an experienced IDEF modeler and not just a trained teacher." The class should be "...full of hands-on case study work — in the METHOD, not just one or more of the automated tools that support the method."[Cullinane et al. 89] We agree with Cullinane's observation that an IDEF class should be taught by an experienced IDEF modeler and not just a trained instructor. One of the attributes that the experienced IDEF modeler brings to the classroom is a motivational attitude. The particular attitude that appears most relevant to IDEF practice is a belief in the effectiveness of team efforts in solving complex problems. An IDEF modeler who does not subscribe to such a belief will generally be ineffective. This is one aspect of the ITS research that we have discovered no mechanism to duplicate. In fact, the current focus of ITS technology is primarily on isolated individual instruction. An area for future research would be development of techniques for simultaneous interactive ITS sessions. It is difficult to effectively simulate a team effort for solving a complex problem with a single individual.

Cullinane et al. identify three levels of modeling proficiency:

- trainee,
- novice modeler, and
- expert modeler.

3.3.2 The Trainee Level

A trainee level modeler knows the discipline component of the method, the language or syntax of the method, and will understand the basic principles of the method. These principles include the mechanics of the author/reader cycle, top-down decomposition and its application, and the interactive nature of modeling.

Obstacles to obtaining the trainee level include the following difficulties which are generally correctable by experience: [Cullinane et al. 89]

- difficulty in seeing multiple potential decompositions other than the first one produced,
- tendency to make diagrams overly-complex or overly-simple,
- inappropriate clustering of arrows,
- tendency to decompose by type,
- tendency to show implementation details in a functional model,
- tendency to corrupt the model with their own bias in terms of knowledge and viewpoint,
- tendency to indiscriminately put everything in the model disregarding its relationship to the objectives, scope, and viewpoint,
- tendency to use a depth first development technique instead of a breadth first top-down approach,
- failure to realize the global impact of a change on the model,
- tendency to include inappropriate details about the decompositions instead of focusing on the interactions among the boxes on the diagram.

Delivering instruction from an IDEF ITS that addresses these issues requires considerable "analysis" capabilities. That is, the tool must actually be able to interpret an IDEF model created by the student for a particular situation. Unrestricted interpretation of this requirement would lead to the conclusion that a CYC class common sense knowledge base and an extremely sophisticated natural language processing capability be present. Based upon discussion with Major Paul Condit, we believe that many of these issues could be effectively addressed by having a library of expert-developed diagrams available with various combinations of the pathological problems identified above. The student could be presented with these diagrams

and asked to rate the similarity of his model to the reference models. Based on the ranking, the ITS could provide feedback pertaining to "possible" pathologies.

In our experimental software, we designed an interface to aid the trainee in getting to the point of actually creating a model. Some of the problems we have noticed occur before the student has the opportunity to make the mistakes described by Cullinane et al. for the trainee level. For example, we have noticed that students tend to have difficulty abstracting the activities and concepts from the source material needed to create the activity and concepts lists. Although they seem to understand concepts and activities, once they are handed a stack of typical source material items, they seem to have trouble initiating the modeling process.

Our experimental software addressed the collection of possible activities and concepts from source material by having the student browse through source material in an environment where the possible concepts and activities are mouse sensitive. The student collects a list of possible activities and a list of possible concepts from the source material. These lists are used later in the model building process as justification for activities and concepts in the model. This approach smoothes the transition from theory to practice and allows the student to walk through the process with guidance when needed.

3.3.3 The Novice Modeler Level

Learning to model requires hands on experience. "Modelers learn best (and most efficiently) when under the guidance of an expert" and the "first project should be of medium to significant size (possibly lasting several weeks)"[Cullinane et al. 89]. Guidance and feedback from an expert modeler is very important. We have also observed that at least one moderate size modeling project is required to advance from the trainee level to the novice level.

After the first project, for example, the novice $IDEF_0$ modeler should, in general, have the following capabilities:

- perfect syntax usage,
- perfect execution of the author/reader cycle,
- ability to abstract $IDEF_0$ activities from process descriptions,
- starting to recognize inappropriate implementation information represented in a functional model,
- ability to identify relation forming concepts,
- starting to anticipate the global implications of a change to a model,
- ability to generate good glossary and text,
- ability to negotiate group concensus based around an evolving $IDEF_0$ model.

Novice modelers often still face the following difficulties which are correctable by experience:

- establishing the scope or boundary of a model (e.g. knowing what to include or exclude in the model),
- determining when to stop decomposing,
- determination of which model or models to build.

We have also noted that novice modelers

- can hold to either an AS-IS or TO-BE perspective consistently.

In our experimental software, we concentrated on ways of developing the cognitive skills needed for IDEF modeling. This included language, observation, and decomposition skills. Development of these skills is critical to solving the typical problems experienced by novice modelers. Primarily, the experimental software takes the approach that practice makes perfect. By providing students with an environment in which they can build a model from start to finish and then compare with an expert's model, the students are given the opportunity to gain experience without the hard knocks. Issues such as viewpoint can be taught more effectively by allowing the students to make mistakes before flagging their error.

One of the more difficult areas to train the student in is the author/reader cycle. It is not so difficult to explain the cycle and its importance, but it is difficult to enhance a person's diligence in keeping the cycle going while showing consideration for others. On the other hand, proper use of the author/reader cycle is critical to successful modeling efforts. Hopefully, the new modeler will already have strong interpersonal skills. If the modeler does not have strong interpersonal skills, other training courses may be required before the student will be able to model effectively.

3.3.4 The Expert Modeler Level

Expert modelers in $IDEF_0$ are generally characterized by the following qualities:

- ability to clearly state the objectives of the modeling effort,
- ability to select the appropriate types of models to build in order to meet the objectives,
- correct selection of the model's scope,
- correct selection of the model's viewpoint,
- ability to construct a model which meets the objectives,

- ability to develop appropriate standards/guidelines to meet the project objectives,
- ability to determine inclusion/exclusion of items based on objectives,
- ability to determine how generic a model should be to meet objectives,
- ability to build models that communicate clearly,
- ability to anticipate the global impact of changes,
- ability to recognize multiple occurrences of the same functions and develop an appropriate representation,
- discernment between organizational partitioning and functional partitioning,
- discernment between decomposition by type and functional decomposition,
- ability to merge models,
- discernment between implementation partitioning and functional partitioning,
- knows when to use FEOs to illustrate a point as opposed to developing a complete scenario,
- recognition of problems and the appropriate corrective action.

Many of these qualities, such as the ability to clearly state the model objectives, require skills which are outside the scope of the IDEF ITS. In the case of stating objectives, the modeler must have the verbal and writing skills necessary to clearly communicate ideas to others. An ITS would only be able to stress the importance of clear objectives.

3.4 Understanding the IDEF$_1$ User

Several general observations can be made about problems that IDEF$_1$ students display based on our experience as instructors and modelers. These include:

- the inability to distinguish objects and their properties from the information actually managed about the objects,
- the tendency to perform model refinement without data to justify refinements,
- the tendency to labor over a single modeling decision rather than making an educated guess and letting the method sort it out,
- the tendency to confuse class and member notions,
- the tendency to not add glossary or descriptions, and
- the tendency to erroneously perform key class migration.

The ITS should be able to address these problems through lessons which describe the problems and practice modeling sessions which reinforce the lessons. The practice sessions will need to address modeling problems which cause students to confuse objects with information kept about the objects. For instance, after presenting source material about a payroll system which does not keep track of the seniority of employees, the ITS could ask the student to add a constraint that employees will receive a $100 bonus on their tenth anniversary with the company. Many students may not look to the source material to see whether the information system keeps track of seniority and just write down a constraint like:

> "For all employees, if it is the employee's tenth anniversary, add a $100 bonus to their paycheck."

or, if they know the Information Systems Constraint Language (ISyCL) [Decker and Mayer 90]:

```
for_all emp of Employee
    if ten_year_anniversary?(emp)
        add_bonus_to_paycheck(emp, 100[dollars]);
```

It may be quite simple for them to write the natural language constraint without considering if the information is there to determine if it is the employee's tenth anniversary. The ISyCL constraint may cause them to think a bit more about whether the information is there or not. Either way, the correct answer would be that the system cannot automatically add the bonus since there is no evidence in the source material that the employee's starting date is available.

Information modeling is more difficult to do correctly than activity modeling. It is easier to picture activities in one's mind along with the flows between them. Information modeling is more complex in that it requires modeling of abstract objects like data records and the relationships between them. Each entity class must be able to be uniquely identified by a key class and these key classes migrate between entity classes due to relationships. The migration of the key classes shows important information and aids in checking the validity of relationships, but the migration is often done incorrectly.

3.5 Pedagogical Levels

Human IDEF instructors are able to adapt the level of their lessons to range from introductory one hour seminars to intense semester long courses. The human instructor must do this to meet the various learning needs of the students. Typical learning needs range from merely a broad introduction of a topic to a very detailed study. A successful ITS should possess the capability to adapt to the learning needs of its students.

The key to the ITS adapting to the student is two-fold. First, each pedagogical level is represented by a different conceptual dependency graph (CDG). Since each CDG represents a course, more work is necessary up front to develop a separate course for each pedagogical level. In Section 3.6, CDGs will be discussed along with the ways in which lessons may be reused to facilitate this course development. The second way the ITS adapts to the student is by varying the instructional strategy used for the individual lesson within the CDG structure.

Pedagogical levels should, by necessity, vary from one ITS to another. The IDEF ITS architecture (described in Chapter 4) allows courses to be planned with any number of pedagogical levels. The following pedagogical levels have been identified during our research based on the suggestions of expert modelers/instructors and the cognitive skills identified for IDEF modeling .

1. Management Overview Level

   This level should be equivalent to introductory seminar type knowledge. Managers and other people who do not necessarily need to model but do need to know general information about the techniques and concepts should find this level sufficient. Please see Figure 3.1.

   None of the cognitive skills identified for IDEF modeling are developed at this level. This level utilizes only CAI in contrast to the other levels which employ intelligent tutoring techniques. Finally, no testing is done at this level and "slide-show" type presentations are acceptable.

2. Reader Level

   This level is targeted at delivery of skills necessary to provide a student with IDEF modeling capabilities similar to those characterized by Cullinane et al. as trainee level performance. It should provide good exposure to the method, the language or syntax of the method, an understanding of the basic principles of the method including the mechanics of the author/reader cycle, top-down decomposition and its application, and the interactive nature of modeling.

   Observation skills and competence type language skills are developed at this level. These cognitive skills represent the lowest level of understanding of our target cognitive skills.

3. Author Level

   This level is targeted at delivery of skills necessary to provide a student with IDEF modeling capabilities similar to those characterized by Cullinane et al. as novice modeler performance. Completion of this level should require perfect syntax usage and execution of the author/reader

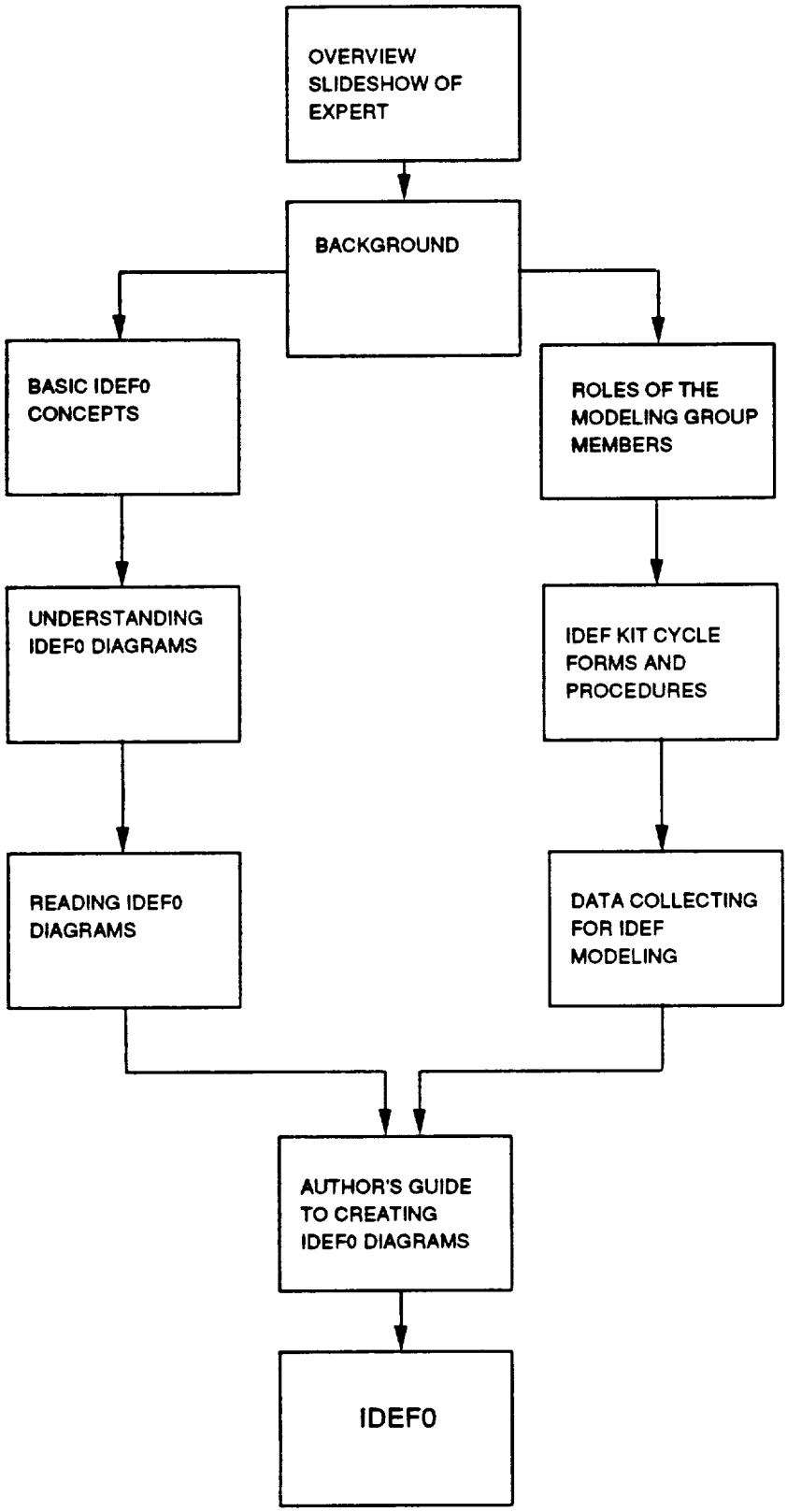


Figure 3.1 Management Overview Level for IDEF0

cycle, proper use of the subject area terminology, ability to create a good AS-IS model, recognition of decomposition by type, recognition of inappropriate implementation represented in a functional model, ability to see common functions, ability to implement mechanism call when the lead consultant points out an application, anticipation of the global implications of a change to a model, and good generation of glossary and text.

All of the remaining cognitive skills identified for IDEF modeling are developed at this level: classification, abstraction, decomposition, and performance language skills. These represent the highest level of understanding of our target cognitive skills.

3. Advanced Applications Level

This level is targeted at delivery of skills necessary to provide a student with IDEF modeling capabilities similar to those characterized by Cullinane et al. as expert modeler performance. It should be similar to the author level but provide more exposure to advanced modeling situations.

The cognitive skills of primary interest at this level are advanced development of abstraction, decomposition, and performance language skills.

As will be discussed in Chapter 6, the experimental software does not address any of these levels directly. Instead the experimental software concentrates on strategies for development of the cognitive skills necessary to meet these pedagogical levels.

3.6 Concept Dependency Graphs (CDGs)

3.6.1 Background and Motivation

Current CAI authoring systems provide a language and usually an environment to facilitate development. Each CAI system developed is generally stand alone in nature. An inherent problem with this approach is that it has a totally "hard coded" nature. We are recommending an ITS architecture that allows for delaying the decision of the sequence of lesson presentation.

By delaying the decision of which hard coded elements of the lesson are used, several benefits should be gained. First, by creating a map of available lessons and what other lessons must be taught beforehand, more knowledge is explicitly available for the tutoring system to make decisions. Secondly, by explicitly creating lessons using different instructional strategies and maintaining that information, the tutoring system again has more knowledge to exploit. Thirdly, by allowing the tutoring

system to maintain control, the tutoring system can exploit the new knowledge in a more dynamic, flexible manner.

A tutoring system must have some representation of the domain knowledge it will be manipulating. By representing an abstraction of the domain knowledge in the form of a network, to be called a Concept Dependency Graph (or CDG), dependencies may be determined in an automated fashion. Such conceptual dependencies are useful for several reasons:

1) a CDG constrains the system to teaching only those lessons for which the student has learned the appropriate background concepts,

2) if a student's understanding of a concept is deficient, the system should systematically administer subtests for each of the concepts that the current concept depends on to pinpoint "knowledge gaps", and

3) by maintaining a short explanation of a concept's importance, a "why" facility may be implemented. By linking together the explanations from each concept in the path from the current concept to the goal concept, a fairly complete explanation of a concept's importance may be automatically generated relative to the goal concept.

The elements of the IDEF ITS architecture which facilitate representation of domain knowledge are discussed in Chapter 4.

3.6.2 Sources of Information for Building Concept Dependency Graphs

In order to test out the notion of concept dependencies described above, we set out to develop $IDEF_0$ and $IDEF_1$ Concept Dependency Graphs (CDGs) manually. The $IDEF_1$ CDG was developed by identifying each concept from the available information and then establishing dependency relationships if possible between the concepts. This task was quite time consuming but provided a worst case baseline for the amount of work involved for the development of a CDG and the difficulty in establishing the dependency relationships. Essentially, a global approach was taken for the identification and classification of the concepts. For the $IDEF_0$ CDG, the provided structure of the IDEF0 manual was used to provide the initial dependency relationships. A local approach was taken in which various major concept areas were identified and their subconcepts were identified. Unlike the $IDEF_1$ CDG, only dependency relationships were established among the subconcepts for each major concept instead of globally establishing dependency relationships among all of the possible subconcepts. Both types of CDGs are useful although the top down approach used for $IDEF_0$ is characterized by a substantially faster development time.

The primary sources of information for isolating the conceptual dependencies in $IDEF_0$ and $IDEF_1$ were the U.S.A.F. manuals. The manuals define the methodologies and provide the essential information needed to develop the CDGs.

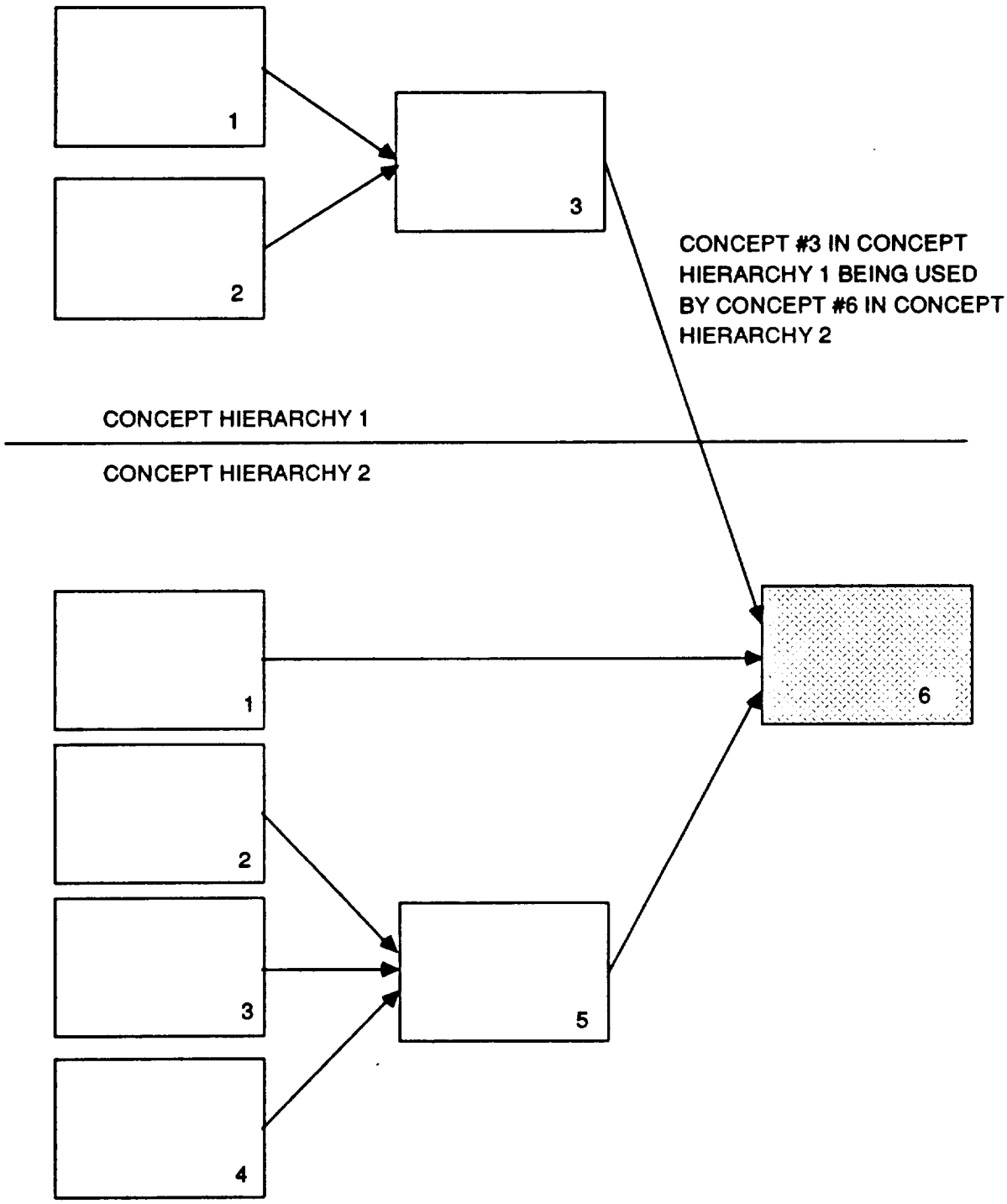


Figure 3.2 Integration across CDGs

Additional information was obtained from experienced users to supplement the manuals.

In general, CDGs will be developed by one of two ways. A top down approach may be used if a course already exists and is therefore already organized. A bottom up approach must be used for developing a course from scratch. All concepts must be identified and dependency relationships established. Levels may then be formed.

3.6.3 Integration Across Concept Dependency Graphs

The architecture for the entire IDEF ITS has been designed for flexibility. This structure enables the IDEF tutor to integrate lessons from other courses where each course is represented by a CDG. The reusable components allow not only the lessons to be integrated but also the files needed for other types of knowledge based information. Please refer to Figure 3.2.

A course developed for $IDEF_1$, for example, should not need to duplicate information that it has in common with a course for $IDEF_0$. The benefit of this becomes more clear, in a general sense, if one imagines twenty interrelated courses already developed. The author of a new course has a foundation to build on in terms of background examples, explanations, and testing. This feature becomes especially useful when a user is putting together a briefing. New lessons may also be developed and added to the existing base at any time. The next chapter describes this integration in much more detail.

3.7 Conclusions

The focus of this chapter has been in several areas ranging from the problems with ITS applied to IDEF systems engineering methodologies to understanding typical $IDEF_0$ and $IDEF_1$ users to the notion of conceptual dependency graphs (CDGs). The most important concepts to abstract from this chapter are the ideas concerning pedagogical levels and the use of CDGs to represent pedagogical levels. It is also important to note the reliance upon the experience of expert modelers to provide indicators for the evaluation and classification of a student in a particular pedagogical level. The pedagogical levels identified for IDEF modeling are not "set in stone." Different courses will require the identification of appropriate pedagogical levels and the corresponding CDGs. The understanding of the CDG concept is crucial to the understanding of the IDEF ITS.

4

IDEF Tutor Architecture and Components

A four module architecture was proposed as a general purpose ITS architecture for structured domains at AAAI '87. It was suggested that there are four main modules of an ITS [Soloway & VanLehn 87]:

- Environmental Module — handles user interface responsibilities,
- Expert Module — provides an expert system for the task domain,
- Student Modeling Module — responsible for the model of the student's knowledge and has expert system capabilities for diagnosis and troubleshooting, and
- Tutor Module — decides what instruction will be given to the student.

While such an architecture appears appropriate for an ITS addressing structured domains, for an intelligent IDEF tutor, that architecture will need to be modified. No known expert system for IDEF modeling exists due to the unstructured nature of the domain. Furthermore, the construction of a good expert system for IDEF modeling does not appear to be currently feasible. There are, however, several ways that the tutoring of the IDEF methods may be automated and "intelligent."

First, instead of trying to capture the IDEF domain knowledge explicitly and creating an internal model, an IDEF tutoring system would need to focus on a set of constrained situations for which internal models could be built. The internal models would capture the "art" that is inherent in IDEF modeling that can only come from expert modelers. By necessity, the internal models would be constrained to situations in which such captured expertise could be exploited in instructional and testing situations with the student.

Second, the curriculum itself may be dynamically manipulated to adapt to the needs of the student. Based on the tutor system's evaluation of the student's knowledge, an intelligent IDEF tutor would avoid teaching concepts which the student already understands. A major weakness in CAI systems is the "one curriculum fits all" approach in which all students see the same information in the same order.

Third, an IDEF tutor should support the capability to vary the instructional strategy. Even in structured domains where it is easier to represent the curriculum knowledge,

it would be difficult to generate lessons dynamically from scratch. This problem is even more difficult for unstructured domains. An IDEF tutor can, however, make decisions concerning when to present a lesson that uses a specific instructional strategy. To accomplish this, the student modeling module must be proficient in its evaluation of the student's current level of knowledge and understanding.

Although it is not feasible for an IDEF tutor to generate lessons dynamically (much less to generate lessons using various instructional strategies), it should be able determine which instructional strategies to use based on its evaluation of the student's responses.

Fourth, an IDEF tutor should support the dynamic integration of multiple domains.

> **Dynamic integration of multiple domains —**
>
> allows an ITS to decide at run time not only which lessons to use from the current course, but also which, if any, lessons from other courses to use to effectively communicate the concepts to the student.

The IDEF family of methods includes $IDEF_0$, $IDEF_1$, $IDEF_{1x}$, $IDEF_2$, $IDEF_3$, $IDEF_4$, $IDEF_5$, and $IDEF_6$. Also, other methods such as ER, data flow diagrams, and structure charts are often used in conjunction with the IDEF methods. To build tutor systems from scratch for each of the above methods would not only be more expensive but would fail to provide a mechanism for one tutor to exploit the effort expended for any of the other tutoring systems. An architecture is needed that will permit, for example, a tutor for $IDEF_0$ to use any lesson available to any of the other IDEF methods. An architecture that permits "reusable" lessons will allow IDEF tutors to take advantage of the common components of the various methods.

A major drive behind the proposed changes to the traditional architecture is the difference in assumptions between tutoring support for instructional domains (systems analysis and design) and more traditional task oriented domains. The AAAI '87 architecture is apparently designed to handle a single domain at a time. The proposed intelligent IDEF tutor architecture handles multiple domains and the dynamic integration of multiple domains. This is extremely important since the IDEFs form a family of methods. Several key commonalities between the methods must be identified including:

- common information,
- common procedures, and
- common concepts and applications.

4.1 Reusable Components

Building ITS/ICAI systems can be very expensive. The cost of custom built tutoring systems costs is high not only due to development but also due to maintenance. Authoring systems are currently available to facilitate development of CAI level software, but the systems developed are essentially "hard coded" lessons that offer little or nothing in the way of dynamic curriculum manipulation or varying instructional strategies. A flexible system which delays the "hard coded" parts of lessons until the last possible moment is a major step in the right direction. Development of tutors for the IDEF family is a good example of how reusable lessons and multiple domain integration would cut both development time and costs. If separate tutors were developed for each of the IDEF methods, just imagine how difficult it would be to change a common set of lessons used by each tutor or to add new lessons after the systems had been deployed. If reusable lessons and a system to manage the lesson relationships are used, all that is required is to send a single copy of each new lesson and an update file for the system librarian (described in Section 4.2.8).

4.2 IDEF Tutor Architecture

In this section, the components of the IDEF tutor architecture are described in detail. Several components have been added to the more traditional ITS architecture to increase the flexibility of the ITS needed for an IDEF tutor. Also, several of the traditional ITS architecture components have been split into multiple components. Emphasis has been placed on generic construction, reusability, and flexibility. Please see Figure 4.1.

The IDEF tutor architecture introduces several new components which distinguish it from other systems:

- *Concept Dependency Graph (CDG).* The Concept Dependency Graph provides a means of representing the curriculum knowledge for lesson presentation and testing. With it, our ideas of dynamic curriculum manipulation and varying instructional strategy become realistic.
- *System Librarian.* Closely tied to the CDG is the system librarian which is needed to manage the complexity of the integration of multiple courses.
- *Model Builder Environment (MBE).* The model builder environment provides a means of developing the cognitive skills of the student needed for IDEF modeling.

The components of the IDEF tutor architecture will now be described in detail.

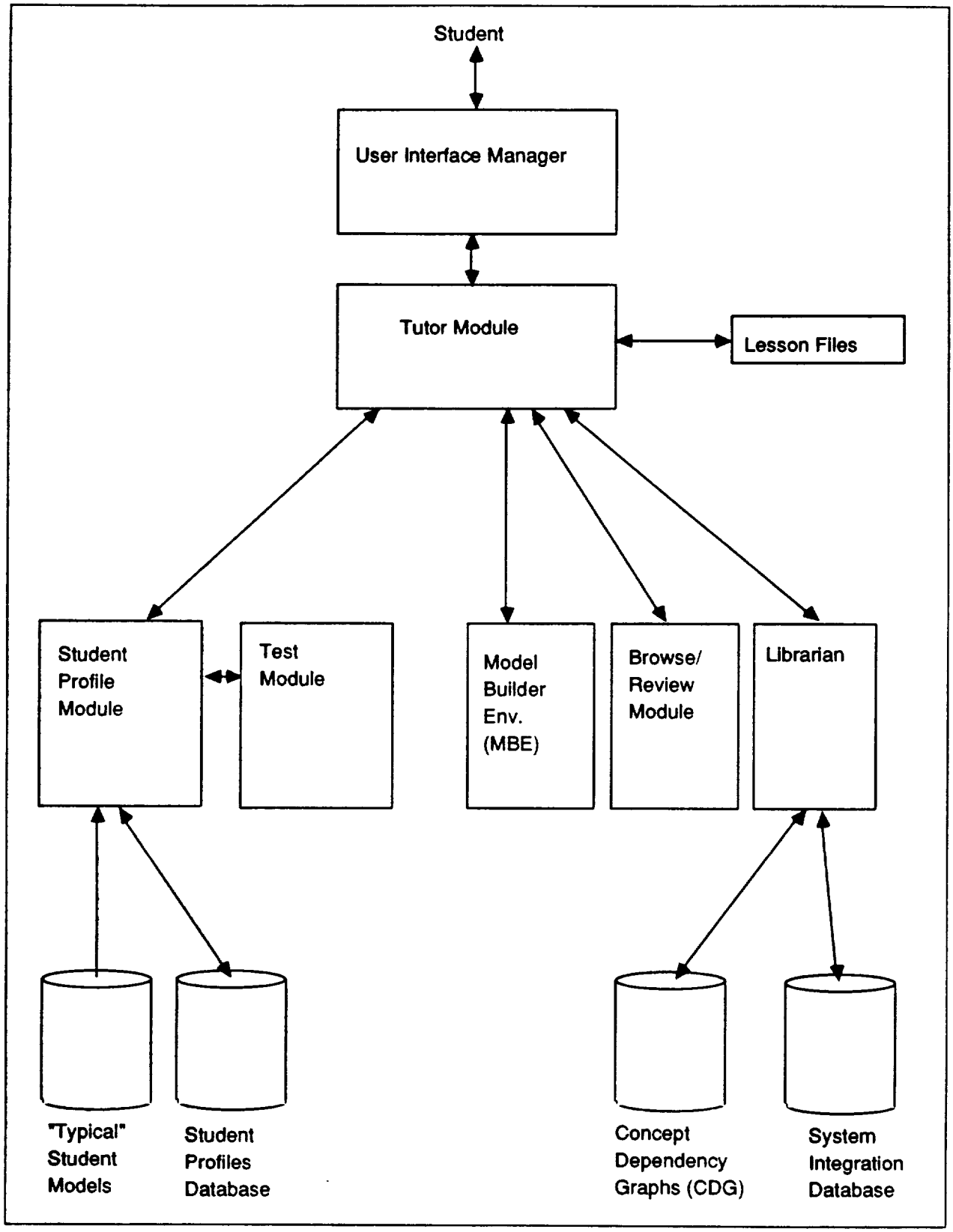


Figure 4.1 IDEF Tutor Architecture

4.2.1 User Interface Module

The user interface module determines the look and feel of the tutor environment. It acts as the interface between the tutor module and the student. This module, like most other modules, is an independent and reusable component designed to facilitate maintainability of the ITS and portability to other systems. The primary goal of this module is to provide the necessary functionality for the presentation of the lessons and the development of the target cognitive skills.

Various human factors issues must be addressed in the design of this module. For example, what is an effective interface for the development of cognitive skills? In what ways should the tutor/student interaction occur?

4.2.2 Student Model Module

Maintaining information about each student is essential for the system to utilize information such as the student's learning goals, modeling experience, and progress on previous lessons. Just as a human instructor builds an implicit mental picture of a student's ability and knowledge, a computer-based tutor must build a model (or models) of the student.

Acquisition of initial information about the student can be accomplished by questioning the student directly. The initial information consists of the student's background in modeling, occupation, and learning goals. Other information could also be gathered to help customize the tutor's interaction with the student. Alternatively, a supervisor of the tutor system could setup the information for each student. With this information, an initial model of the student is formed. The student modeling module will continually use and update the model of the student based on the interaction with the student.

The student modeling module also keeps "typical" models of students for classification purposes. These models are needed to compare the current student's model against for classification of the student. It is important for the student modeling module to have "good" as well as "bad" generic models with which to compare the student against. This enables the identification of acceptable as well as unacceptable learning patterns.

Additionally, the student models will provide data for research purposes and improvement of future systems. It is important to identify poorly designed, ineffective lessons to improve future systems. By the same token, it is important to identify learning bottlenecks for students.

4.2.3 Concept Dependency Graph (CDG)

The Concept Dependency Graph (CDG) provides an overall multi-level framework of the curriculum. It is how the curriculum knowledge is represented for manipula-

tion by the tutor. The CDG explicitly establishes dependency relationships between lessons. This dependency information is exploited for curriculum guidance and lesson delivery as well as for testing. All of the components of the tutor architecture, except for the CDG and the lesson files, are constructed in a generic fashion. All of the information that distinguishes courses is maintained in the CDG and the lesson files for the individual lessons. Developing a new course would consist of developing the CDG and its related files and then updating the system librarian.

The key to referencing across CDGs depends on each CDG essentially being stand alone in nature. No CDG should directly reference how it is mapped into other CDGs. An approach that permitted CDGs to directly reference other CDGs would create inflexible, "hard coded" dependencies. Every time a new course was created and released, other courses would potentially require updating to reflect any new mappings. By using a system librarian to maintain and inform the tutor module of interlesson mappings, the system can be integrated across courses in a very flexible yet generic manner.

CDGs have been developed for $IDEF_0$ and $IDEF_1$. The $IDEF_0$ CDG consists of Levels I (management overview level) and II (reader level) and was developed using a top-down approach. The $IDEF_1$ CDG consists of Level III (author level). They were constructed to gain an understanding of the complexity involved in isolating the concepts in $IDEF_0$ and $IDEF_1$ and for establishing commonality between the two for later use. Isolation of the common concepts is important to conduct any experimentation with integration of CDGs.

In the Level III $IDEF_1$ CDG, an attempt was made to remove redundancy. If concept "A" already preceded concept "B" indirectly in the graph, then a direct path from concept "A" to concept "B" would be considered redundant and eliminated. This simplified the CDG routing considerably. In a computer based implementation, however, all dependencies would need to be maintained to facilitate usage of concepts in one CDG by another CDG.

Although the CDGs should eventually reside inside databases, they are currently on paper in the form of large diagrams. This format provides a means for grasping the overall picture of $IDEF_1$ and $IDEF_0$ when broken down into its individual concepts during the development process.

Flexibility is maintained since the concept dependency database maintains the course description at a conceptual level. By rearranging links in the database, the entire Concept Dependency Graph – and therefore the course itself – may be reorganized. Reorganizing the CDG should be invisible to the other modules (e.g. the tutor module, the test editor, the review module, and the browser) due to their independence from the database implementation.

4.2.4 Tutor Module

The tutor module is responsible for actually instructing the student. This involves interaction with the system librarian for access to the Concept Dependency Graph for curriculum guidance, the lesson files for content, the system librarian for system integration, the student model module for evaluation and pedagogical recommendations for the student, the review module, and minor subsystems.

The tutor module must:

- request the next lesson from the system librarian,
- conduct the next lesson session,
- request testing and evaluation of student's understanding of a specific concept area from the test module, and
- request evaluation of student's progress from the student model module,

The tutor system uses two levels of diagnostic evaluations. First is the diagnostic evaluation of a student's understanding of a specific concept area. The test module, which is described in Section 4.2.5, performs the diagnostic evaluation of the student's understanding of a particular area. At a higher level, the student model must reflect the student's overall understanding of the IDEF method. Conclusions about overall understanding will depend on the diagnostic evaluations of specific concept areas, but should not simply be a summary of the evaluations. Pedagogical strategy decisions should be based on this higher level evaluation.

4.2.5 Test Module

The test module is responsible for conducting all testing before and after a lesson. Although traditional types of CAI testing procedures should be supported, the test module for an intelligent IDEF tutor will, by necessity, have higher demands. The test module must be able to intelligently diagnose problems or potential problems that the student has or develops. This entails testing the development of the identified cognitive skills for IDEF modeling.

The IDEF tutor should concentrate on capturing the "art" of IDEF modeling. The approach taken for testing should remain consistent with that focus. The test module must, therefore, support the same type of internal models for specific situations that are used by the tutor module.

A student could be asked, for example, to select possible activities from source material. The test module compares the choices made by the student with the internal model. The student's incorrect choices can be used to help identify misconceptions. If the suspected problem area can be decomposed, the test module should attempt to isolate the suspected problem to the finest level of granularity by repeating the test

process with a new internal model and set of questions. Once the problem areas are identified, the test module notifies the tutor module of the need for further tutoring.

Another example of utilizing an internal model for comparison with the student's work involves the confusion between objects and the information maintained about objects. A typical warning signal of this problem is the tendency to pluralize entity class names. If the student exhibits this tendency, a match would be made against the internal model revealing a problem area.

Repairing the suspected problems should be conducted in a manner similar to the way that they were found. Expert modelers/instructors often devise their own set of heuristics to isolate and correct mistakes. It is not unusual for modelers to form their own collection of test cases based on experience.

4.2.6 Browser/Review Module

A user may wish to browse a set of concepts without engaging the tutoring process. A hypertext style browsing capability of specific topics should be available starting from an index or table of contents. Alternatively, a CDG could be browsed. Please see Figure 4.2.

Review is a key component in learning. The student will generally review before a lesson or test. The student selects the topics to be reviewed and is provided with a hypertext style review of those particular topics. The student may review until ready to begin the lesson or test.

4.2.7 Lesson Modules

Each lesson module should essentially carry all of the information necessary to teach the procedural and textual aspect of a lesson. Each concept in the CDG should have an associated lesson. Two approaches have been suggested for implementation of the lesson module. One approach uses the tutor module as an interpreter with the lesson modules as input. A second approach uses the lesson modules as stand alone executable code that communicate with the tutor through messages in files.

Currently, the first approach seems more flexible and efficient. It also avoids the problems of transferring control from module to module. The disadvantage is that the lessons must be written so that the tutor module can interpret them. Currently, the interpretive approach if favored since it allows the tutor module to maintain control. This facilitates communication with other modules.

Two levels of instructional strategy decisions must be managed by the tutor system. The higher level is concerned with strategies for deciding when to use different instructional strategies. The lower level is concerned with handling responses from the student that vary as different instructional strategies are used. The lesson files should be concerned with the lower level of decision making. The higher level must

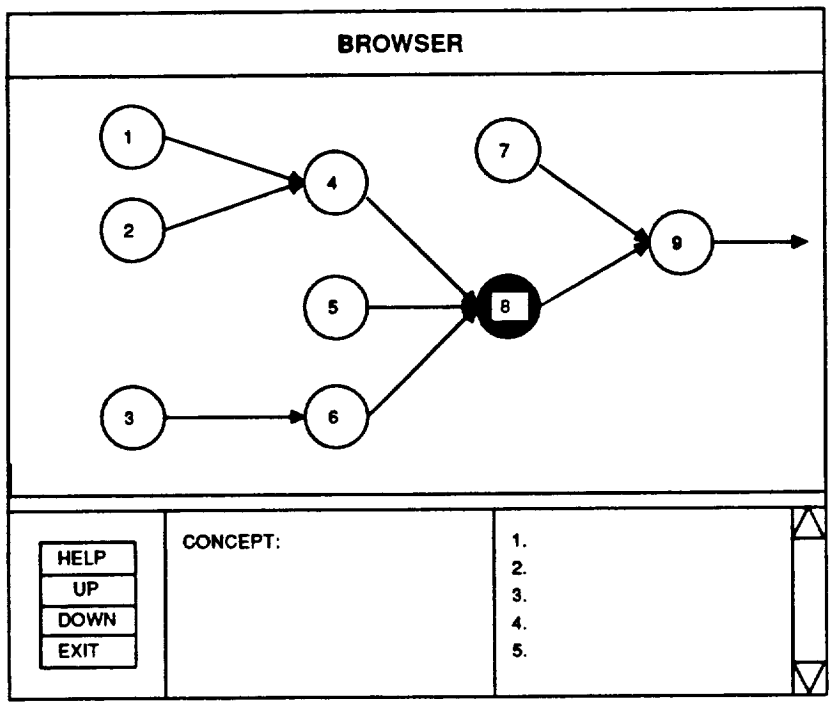


Figure 4.2 Graphical Browser

monitor and evaluate the student's progress to rate the effectiveness of a particular instructional strategy for the specific lesson and the specific student.

The lesson modules provide an ideal place to embed specific instructional strategies. Although the ITS system will determine which instructional strategies to use, the lessons still must be written by a human instructor at the lower level incorporating the different instructional strategies. The key to the system being able to utilize different instructional strategies is the existence of multiple copies of the same lesson, each constructed using a different instructional strategy. The lesson used depends upon which instructional strategy has been selected. Having lessons based on multiple instructional strategies provides highly desirable flexibility. When a student does not understand a lesson taught one way, instead of replaying the same lesson, the system would have the ability to run the same lesson again using a different instructional strategy. Since certain instructional strategies are more appropriate for specific learning situations, the ITS will rely on the higher level instructional stragegy decisions to select an appropriate strategy.

4.2.7.1 Lesson File Example

After describing the purpose and usage of a concept, it is important to reinforce the concept through practice. In our experimental software, we experimented with a

scenario in which the student is given source material and asked to select possible activities and concepts from the text. As the student reads the text and moves the mouse over the text, words and phrases are highlighted. The highlighted items are possible activities or concepts.

In classes at Texas A&M, students learning IDEF often feel "overwhelmed" when given a large stack of source material with instructions to create the activity and concept lists. The scenario used in the experimental software allows a variety of different source material items to be used such as interviews and reports. This approach is designed to guide the student through the information gathering phase during the student's initial exposure to the method, thus enhancing the student's confidence in the procedure. It was determined that it would be better for the student to be allowed to pick any word or phrase instead of just the ones the system had predetermined. Although we initially found it helpful to provide mouse sensitive guidance when selecting possible activities and concepts, the student needs more freedom to make mistakes. It is best to let the student discover a mistake due to failure in latter stages of the modeling process.

4.2.8 System Librarian

To manage the additional complexity of multiple domains, a component of the architecture should be dedicated to that purpose. This component must remain independent of the lessons themselves and merely manage the relationships between the lessons. The tutor module must be able to query this component for the existence of relationships and intermethod references and then use this information to guide the dynamic curriculum manipulation. This management component is called the system librarian.

The key to referencing across CDGs depends on each CDG essentially being stand alone in nature. No CDG should reference any other CDG. If CDGs referenced one another, all of the other courses would potentially need to be updated to reflect any new mappings every time a new course was released. Please see Figure 4.3.

By using the system librarian to maintain and inform the tutor module of interlesson mappings, courses may be developed that use lessons developed for other courses in a flexible and generic manner. The system librarian maintains a database of available lessons and how they may be mapped across Concept Dependency Graphs. Updating the system librarian should update the entire system concerning mappings across courses. The tutor module should query the system librarian for any possible mappings. If a mapping existed but the lesson was not available, the tutor should proceed as if it did not exist. If the mapping existed and was loaded, the tutor should integrate the information making the entire process invisible to the user.

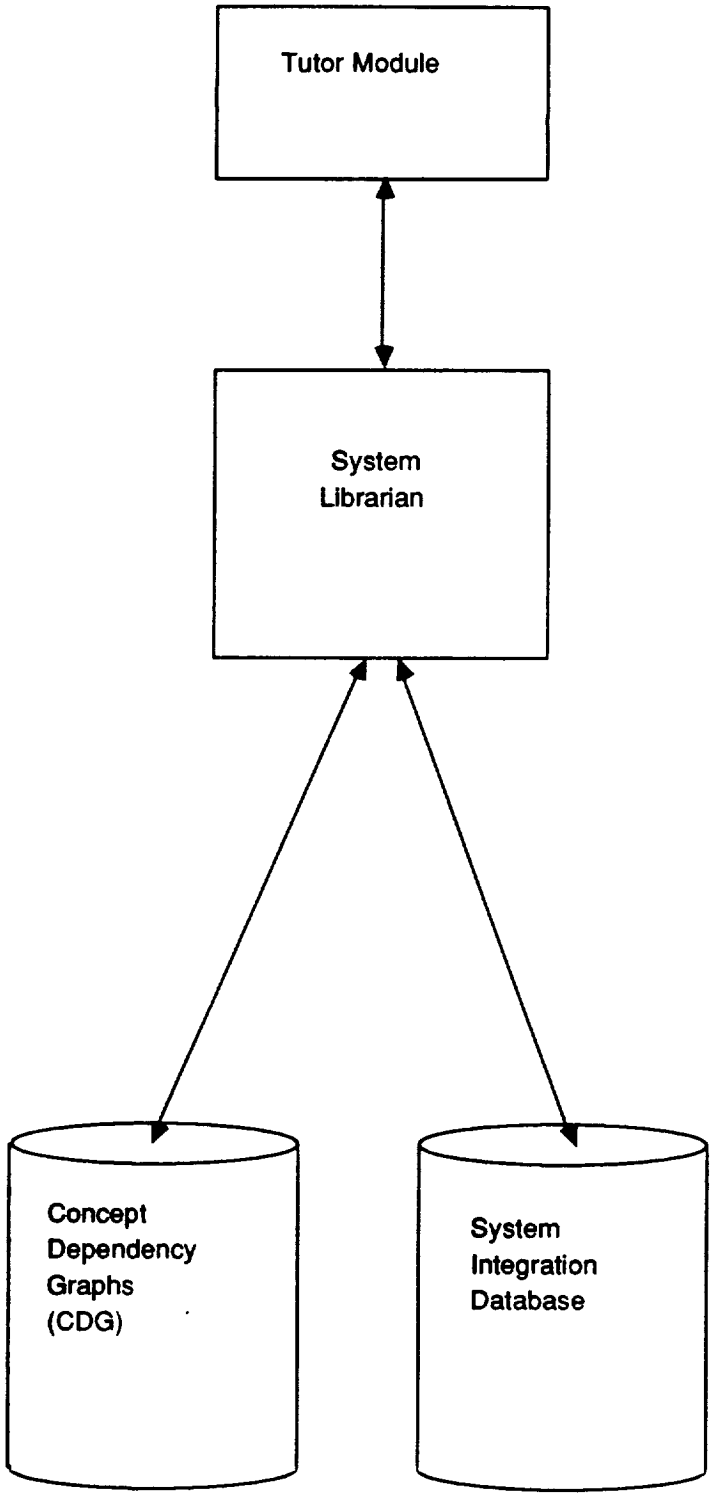
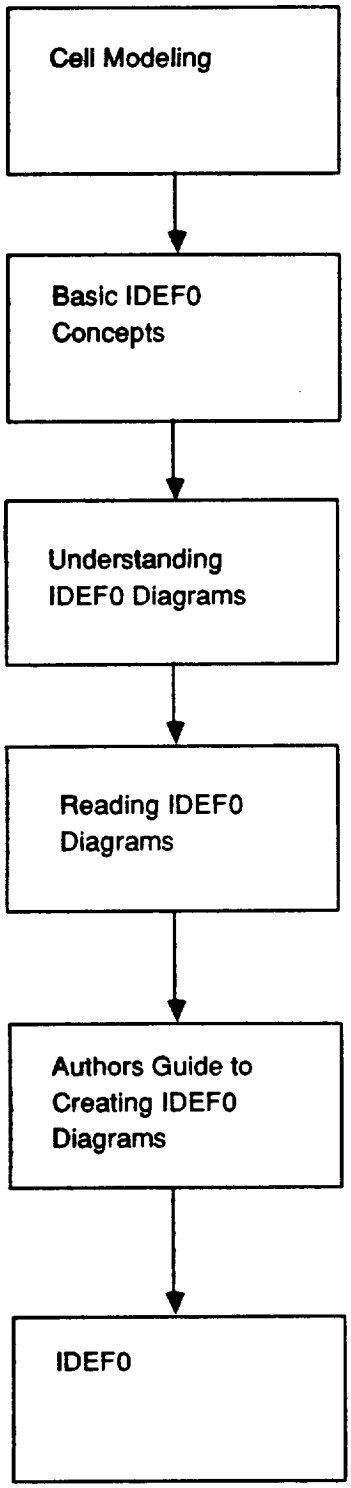


Figure 4.3 System Librarian



The IDEF0 technique and basic concepts are based upon a cell modeling technique know as the structured analysis and design technique (SADT).

The basic IDEF0 concepts provide a conceptual framework for the understanding of IDEF0 diagrams.

Understanding IDEF0 diagrams consists of model definition, the IDEF0 symbols, and the importance of timing and sequence. This provides a background for reading IDEF0 diagrams.

Reading IDEF0 diagrams involves how to approach a model and understanding the semantics of boxes and arrows. Reading diagrams is an essential skill needed before one may proceed to authoring.

Authoring diagrams is the last of the basic skills required to actually be capable of IDEF0 modeling. It involves the basic steps, model creation requirements, drawing diagrams, graphic layout, writing text, model quality checklist, and data collection.

Figure 4.4 Example "Why" facility explanation

4.2.9 Model Builder Environment (MBE)

The difficulty of automating the evaluation of a model as a whole requires much research. The problem of model evaluation also complicates the diagnosis of a student's ability to model. An intelligent IDEF tutor system clearly requires a mechanism to automate model evaluation and diagnosis of the student's current modeling level. The Model Builder Environment (MBE) addresses these issues, and is discussed in Chapter 5.

The use of external modules or tools already in existence should greatly reduce development time and costs. For the IDEF tutor system, the Model Builder Environment (MBE) is considered an example of an external module. By keeping all such model building environments as external modules, the tutor architecture is kept as generic as possible without loss of functionality or extendability.

4.2.10 Minor Subsystems

Human instructors are often asked questions such as "why is this concept important?" Given a CDG, an IDEF ITS can answer that question by tracing the network from the current concept node to the goal concept node. Given that each concept node had a short descriptive text string, the ITS system could form a paragraph describing specifically how the current concept relates to the target concept and the intermediate concepts. The primary advantage of a dynamic "why" facility is in its flexibility. Hard coded complete "why" explanations would be inconsistent if the CDG was reorganized. Please see Figure 4.4.

The "why" facility uses the highest pedagogical levels available for simplicity. By tracing a path from the current module to the top level as quickly as possible and then proceeding at that level to the goal concept, simplicity in the explanations is maintained. The same "why" facility concept could be applied to lower pedagogical levels but the explanations become very long and the point of using the facility tends to become lost in the volume of explanation generated. For example, an explanation consisting of explanations from 100 concepts would discourage most people from using the facility.

4.3 Conclusion

The IDEF tutor architecture was developed based upon the representation scheme selected to represent a course and its associated lessons. This representation scheme is called a conceptual dependency graph (CDG). A major advantage of this representation is the ability to provide dynamic integration of multiple courses or domains. In other words, courses may be easily modified and even built upon the foundation provided by previously constructed courses. Lessons may be easily reused along with their associated testing instructions. A second major advantage is the ease with which courses may be dynamically customized based on the student's response.

Varying the instructional stragegy is actually only an extension of the ability to dynamically customize a course.

The three key components which distinguish the IDEF tutor architecture from other systems are the CDG, the system librarian, and the MBE. The CDG, as just mentioned, provides a means of representing the curriculum at a high conceptual level. The system librarian is necessary to manage the complexity of accessing multiple CDGs, the relationships within each CDG, and the relationships between CDGs. Finally, the MBE provides a special environment for developing the cognitive skills typically necessary to be successful at IDEF modeling.

5

Developing Cognitive Skills for IDEF Modeling

The overall goal of the IDEF Tutor is to teach modeling. Several difficult questions arise when trying to automate such a task. For example, how can a model created by a student be verified and evaluated within a particular context? If experts create different models for the same scenario, how can the quality of each model be determined? The problem is similar to that encountered with natural language understanding. Like the semantics of natural language, the semantics of modeling are difficult to manipulate and evaluate. Given the overall goal of teaching modeling, however, is there another way to achieve the same goal? Could the isolation of the skills used to teach modeling and their development provide a "backdoor" approach to our overall goal? Our research suggests that it could. In particular, the target skills are cognitive skills, and their development could enable an easy transition to our overall goal of teaching modeling.

Knowledge

Comprehension

Applications

Analysis

Synthesis

Evaluation

Figure 5.1 Bloom's Taxonomy [Bloom 56]

Bloom's taxonomy (Figure 5.1) identifies six categories of learning from a cognition perspective which range from simplest at the top to the most complex levels at the bottom. Of the five cognitive skills identified as necessary for IDEF modeling, abstraction, decomposition, and performance type language skills are considered the most complex. Please see Figure 5.2. Our experience with the experimental software has convinced us that observation, classification, and the competence type language skills can be taught through a combination of CAI level lessons and interaction with

Observation

Classification

Abstraction

Decomposition

Language

Figure 5.2 Cognitive Skills for IDEF Modeling

the Model Builder Environment (MBE). Abstraction, decomposition, and performance type language skills, however, require ITS capabilities. The Model Builder Environment Module is specifically designed to develop such cognitive skills.

Our research suggests three approaches to enhance the development of these cognitive skills. The first approach is referred to as the constrained discovery process approach. It attempts to allow the student to build a model under constrained conditions. This approach is the most difficult to develop due to the difficulty of verifying the student's work, and because a choice must be made between analysis of *what* the student models and *how* the student models. Due to this difficulty, specific limitations and design concepts for this approach require further research and experimentation.

The second approach attempts to develop higher order cognitive skills such as analysis, synthesis, and evaluation. This is done using a comparative case study analysis approach using the MBE. The basic approach taken in the MBE is rooted in the lessons we learned with our experimental software in which we experimented with ways to develop the cognitive skills needed by the student. In the experimental software, for example, initial observation skills were developed by providing source material with mouse sensitive activities and concepts.

The third approach is reserved for more experienced students. Referred to as the model library approach, it allows a student to actually create a model and then try to verify its correctness by comparison against models in a library.

The following sections explore the three approaches in more detail along with some conclusions.

5.1 Approach 1: Constrained Discovery Process Approach

To apply the constrained discovery process approach, several key decisions had to be made early in the design process. Due to the difficulty in verifying the student's models a choice had to be made between analysis of what the student models and

how the student models. The constrained discovery approach is based on the decision to analyze how the student models although some attention must be given to what the student models.

By constraining the way the student builds a model, the combinatorial explosion of valid and invalid model variations is avoided. Instead of trying to evaluate the model after it is completed, a diagnoser component of the model builder environment captures the mental states the student proceeds through to create a model. The student's problem solving plan for each stage is then compared against internal models representing different valid and invalid ways the student is allowed to model the scenario at that stage. Incorrect and alternative modeling plans are represented in the internal models for a possible match of problem plans. The modeling scenarios must be carefully selected to minimize the possible permutations of model alternatives. A "critic" that advises in a context sensitive manner provides advice and hints on the student's current choices.

By only allowing the student to proceed down certain modeling plans and by dividing the plans into stages, matching the student's work against previously developed plans is practical. By anticipating as many alternatives as possible and explaining their potential problems, the system eventually allows the student to proceed through only a few carefully developed modeling plans. Each plan is composed of stages with associated subplans. Although this may seem overly restrictive at first, more time is spent by the student interacting with the tutor comparing alternatives and ways to achieve the same goal. The "discovery process" in this context is therefore actually more of a "constrained discovery process" in which the student may make new discoveries, but only within a limited range. The key is to select a range of alternatives which is most beneficial for the students.

5.2 Approach 2: Case Study Analysis Approach

The case study analysis approach never allows the student to actually create a model. Instead a significant amount of time is devoted to analyzing well constructed and poorly constructed models. This approach utilizes carefully prepared case studies, similar to the first approach, but instead of creating models the student analyzes models. The student examines the alternative models and compares the various alternatives. In general, this approach is used before the student is allowed to create models.

The student must learn to distinguish poorly constructed models from well constructed models. By presenting models for a side by side comparison, the student can more readily note differences and relate the models to the context of the situtation being modeled. Please see Figure 5.3. Case 1, for example, may be more appropriate in a certain context than Case 2. The student could be asked to analyze and evaluate the models given various contexts. While subtle differences can be highlighted using

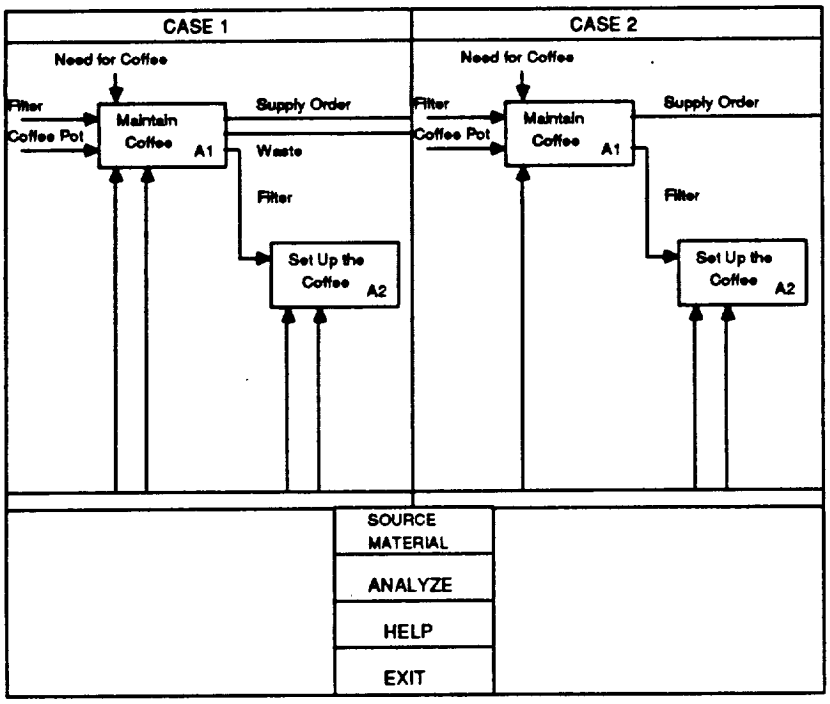


Figure 5.3 Case Study Analysis Approach

this technique, completely different approaches to the same scenario can be shown as well.

A student may be asked to mouse click on areas of a model and then justify the choice from available explanations. A point evaluation of the student's analysis is made and compared against internal models, a problem list is then generated from which a remedial agenda is created. Finally, the student is given feedback and remedial instruction.

Similarly, a more advanced student may be asked to evaluate two models. The student must decide if the models actually represent the situation being modeled and if so which model is better. Both models could be good, both could be incorrect, or one could be more accurate than the other. These activities are designed to facilitate analysis and evaluation type learning.

5.3 Approach 3: Comparison Against Model Library

While the constrained discovery process approach focuses on how the student solves the problem and the case study analysis approach focuses on analysis of models, the model library approach allows the student to create models. The main idea of the

model library approach involves the comparison of a model developed by the student against a library of similar models and diagrams. By shifting some of the comparison and matching tasks to the student, the problem of model comparison is reduced. The development of model library is a key component for such an approach to work. Incidently, the use of a model library could be made available to advanced modelers for reference in work situations.

Using a library of models involves asking the student to choose the model from a set that most closely matches the one constructed by the student. The model library should contain good models as well as poor models. The poor models should exhibit problems common to new modelers as well as poor modeling techniques. The student's selection provides information to the tutoring system which may be difficult to obtain otherwise.

This approach forces the students to analyze and reanalyze their work during comparisons with other potential solution strategies. It was mentioned earlier that a key to effective learning is the implicit knowledge the student gains while learning the explicit knowledge. This implicit knowledge is what actually ties all of the other knowledge together for the student. Gagne calls this new implicit knowledge the "glue" that ties together pieces of knowledge [Gagne 62]. It is important for a student to recognize poorly constructed examples as well as well constructed examples. The goal of this comparative approach is the development of the implicit knowledge by the student. If the student only gains the explicit knowledge, the student is merely memorizing the material and will be no closer to learning how to model than before.

5.4 Conclusions

Three approachs have been identified as appropriate for the development of the target cognitive skills for IDEF modeling. The constrained discovery process approach allows the student to build a model although under highly constrained conditions. The case study analysis approach engages the student in the analysis of carefully prepared case studies forcing the student to examine and compare well constructed and poorly constructed alternatives. Finally, the comparision against model library approach relaxes many of the previous constraints allowing the student to create and comparatively analyze a model against models from a model library. Of the three approaches, the model library approach is expected to cause the most difficulty when automating the validation of the student's work. A combination of these approaches should be used for development of the target cognitve skills for IDEF modeling. All three approaches can be used in the same course if necessary to meet the student's goals.

In the experimental software, we experimented with the construction of $IDEF_0$ models. The constrained discovery approach represents a more constrained variation of that experimentation in which our focus is actually on what steps the student proceeds through to accomplish a goal. The comparison against models in a model

library approach represents the basic approach developed in the experimental software and then extended with the addition of an analysis component.

The development of the target cognitive skills depends heavily on the Model Builder Environment (MBE). The MBE consists of a menu driven environment for building models and a knowledge based system (KBS) for evaluation of the current plan against internal plans. The MBE must be menu driven to constrain the possible mental states that are allowed and facilitate their capture. The MBE can also access the conceptual dependency graph so that it can determine dynamically which rules, constraints, internal models, and internal plans are appropriate for the student's current level of knowledge. It is important for the MBE to adjust its actions dynamically based on the student's progress through the course. This is analogous to a human teacher that not only knows what the student should know based on their progress, but also what the student should not know based on what has not been covered. It would be inappropriate to use a concept in a explanation, for example, if the student has not yet been exposed to that concept.

6

Experimental Software

As part of the exploration into intelligent tutoring systems (ITSs) for the IDEF methods, experimental software has been developed for $IDEF_0$. The purpose of the experimental software is to bring to light issues regarding user interface design and ITS strategies for system engineering methodology training that are difficult to ascertain without hands-on use. The experimental software is not meant to be a finished product, or even a prototype, only a vehicle for evaluating and displaying concepts of importance.

The experimental software concentrates on issues regarding teaching the method processes. In other words, how one goes about building a model. Consequently, the experimental software does not concern itself with issues important in teaching basic syntax and semantics, but instead demonstrates a "practice field" for novice modelers.

The experimental software has been developed on Symbolics LISP Machines. The Symbolics machines provide a powerful environment for experimentation. Besides LISP, LISP Machines provide interface development and debugging tools that are at least on par with, and usually superior to, tools found on other platforms.

Little knowledge of the Symbolics is required to run the experimental system. More than enough background is contained in Appendix D of the "IDSE User's Manual - Version 1" [Wells 88]. To load the IDEF Tutor demo, enter Load System Tutor after logging in. Once loaded, the experimental system is selected by typing SELECT 3.

There are two parts to the experimental software. The first part displays issues regarding ITSs for $IDEF_0$. The second part is the on-line reference for $IDEF_0$. The on-line reference was derived from the standard reference (the "Yellow Book") for $IDEF_0$. The on-line version has many advantages over the paper version. These advantages will be described in this report. The on-line reference should prove useful to anyone requiring knowledge of $IDEF_0$.

6.1 Overview of the Experimental Software

The basis for the experimental software is described in the "ICAI for Systems Analysis Methodologies Technical Report" [KBSL-90-601].

In an actual system, when the student first starts the system he or she would be prompted for identification. If this were the first time the individual had used the system, he or she would be prompted for background information. This information would be used to configure the system for the particular student. Since students may have diverse backgrounds, the system would use background information to customize examples and to determine the pace. The experimental software does not take advantage of background information, thus there is no login procedure.

In order to instruct the student in the modeling processes of the methods, a system could use a combination of "slide-show" and "interactive" sessions. Slide-show sessions allow the student to watch a model constructed correctly. Interactive sessions allow the student to follow the same steps in creating a new model. The experimental system has only the interactive session, since the slide-show session would just be an automated version of the interactive session.

The scenario for the interactive session is based on making coffee. This example was chosen due to its clarity and familiarity to many users. Actual ITS scenarios would probably be based on activities which are more relevant to the backgrounds of the students.

The experimental system is built on top of an enhanced version of the prototype $IDEF_0$ tool developed under the Integrated Information Systems Evolution Environment (IISEE) Project for the Air Force Human Resources Laboratory. Thus, the system provides integration between instruction and modeling. For a description of the original "Model Builder" prototype, see [Wells 88].

The Model Builder included modes for diagram editing, concept editing, and activity tree editing. These modes are also available under the current system.

The first point in any modeling process is the collection of source material. The experimental system takes care of this for the student; providing the source material from which the student is either shown how or asked to select source data items. In this case, possible activities and concepts are identified from the source material. This information gathering phase is the first mode of the experimental system.

The system has two scopes: activity and concept. The scope controls the operation of the system. For instance, in the information gathering mode, if the activity scope is selected, selected source data items are considered to be possible activities.

When selecting source data items, the display shows the source material on the left hand side and the selected source data items on the right side. A message pane at the

bottom right provides feedback to the student and the command pane at the bottom left allows the student to enter commands. A typical information gathering screen is shown in Figure 6.1.

Once the source data items are collected and identified as activities or concepts, it is time to construct the model. There are two available modes for model creation: graphical and textual. The graphical view is simply the normal graphical presentation of IDEF$_0$ models. Different commands are available depending on which scope is selected.

In textual-activity mode, the possible activities and an activity tree are displayed. The activity tree allows a broader view of the model, but no information is displayed other than the activity names and numbers. The activity tree allows hierarchical viewing of the model where lower levels can either be displayed or hidden.

In textual-concept mode, the possible concepts and concepts used in the model are displayed. The subparts and subtypes of each concept are also shown.

Graphical-activity and graphical-concept modes are similar other than which source data items are displayed. Under activity scope, possible activities are shown, whereas under concept scope, possible concepts are shown.

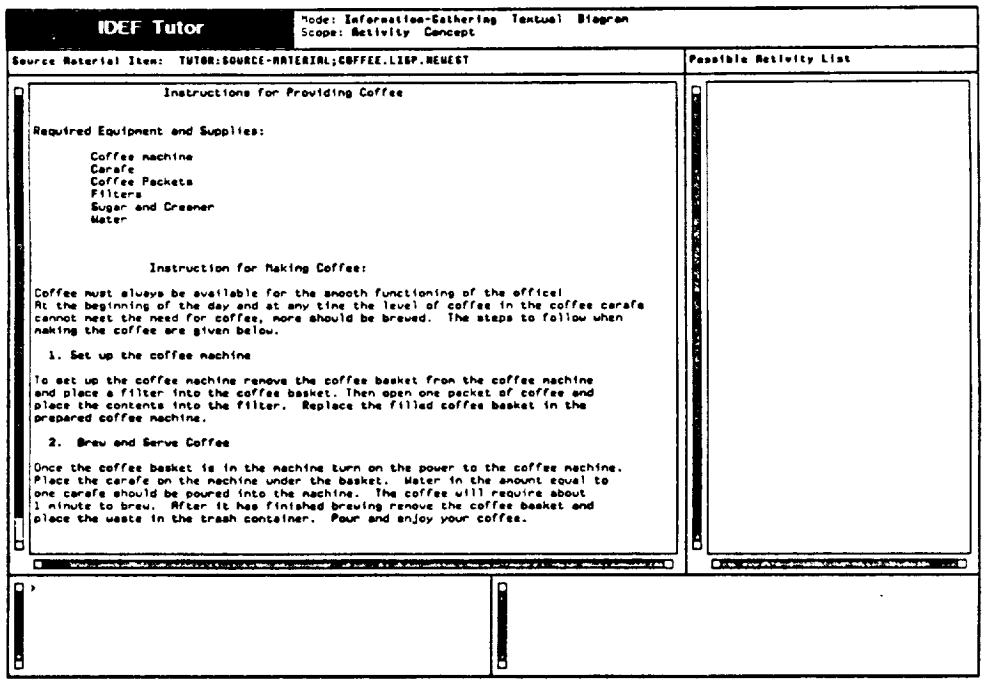


Figure 6.1 Information-Gathering Mode

6.1.1 A Demonstration: An IDEF$_0$ Model of Making Coffee

The following describes the process of using the interactive portion of the IDEF Tutor. If you have not done so already, load the IDEF Tutor by entering "Load System Tutor" from the LISP Listener. Once the system is loaded, type SELECT 3. You should now be in the IDEF Tutor.

To begin, start an interactive session by using the command:

Start Interactive Session {New or Old} <session-name>

This notation has the following meaning. The command name "Start Interactive Session" can optionally be followed either by "New" and the name of a standard session or by "Old" and the name of a session which is in progress. If the space bar is pressed instead of typing "New" or "Old", the system will assume the last choice made, or if the system is fresh it will assume "New".

In this case, choose "New" and then type "Coffee" or select it from the menu generated by clicking right. The Tutor system will now enter interactive mode.

The display should look like Figure 6.1. There are many panes on the display. At the top of the screen next to the title pane is the Mode Selection Pane. There are many modes of operation associated with model creation.

The first mode is the "Information-Gathering" mode. In this mode, source material is used to identify possible source data items. These source data items may be possible activities or possible concepts.

The "scope" identifies what the student is looking for or working with. There are two scopes, "activity" and "concept". The combination of mode and scope give six different modes. In this presentation we will refer to these modes by names such as activity-info-gathering mode, which is the "Information-Gathering" mode with "activity" scope.

Activity-info-gathering mode is the initial mode of operation. Consequently, the first task is to identify possible activities from the source material. The system-supplied source material for making coffee is in a file called "TUTOR:SOURCE-MATERIAL;COFFEE.LISP". The system automatically presents the first source material item associated with a given session (e.g. coffee). In this case there is only one source material item, thus it is displayed upon entrance to the interactive session.

To identify possible activities, read through the text looking for verb phrases which sound like reasonable activity names. The suggested method of browsing the source material is to sweep the mouse cursor across the line as you read. Only certain nouns and verb phrases are mouse sensitive. Verb phrases which are not mouse sensitive cannot be selected as possible activities. In an actual system, there would be a more

sophisticated manner of handling the source material. Such a system would allow the student to identify the activities and concepts without limiting the choices. Later analysis phases would show why choices were either good or bad.

To add an item to the Possible Activity List, click left on the highlighted item. Note that this creates a source data item which may or may not actually be used in the model. Adding activities and concepts to the model happens later in the process. Also note that the system will not allow you to choose the same noun or verb phrase twice, but synonyms and slightly differently spelled variants will be accepted.

Once you have made both passes through the source material, the possible activities and concepts should have been identified. Now it is time to take the possible activities and determine the actual activity hierarchy. To do this, you need to switch to textual-activity mode by clicking on "Textual" at the top of the screen. The screen should look like Figure 6.2.

The modeling process can either be done top-down or bottom-up. In this example, we will work top-down. The first activity to be identified is the "A0" activity. The "A0" activity describes the activity being modeled, in this case, making coffee.

Once you have determined which possible activity is the "A0" activity, you can create the activity by clicking left on the possible activity and then supplying the

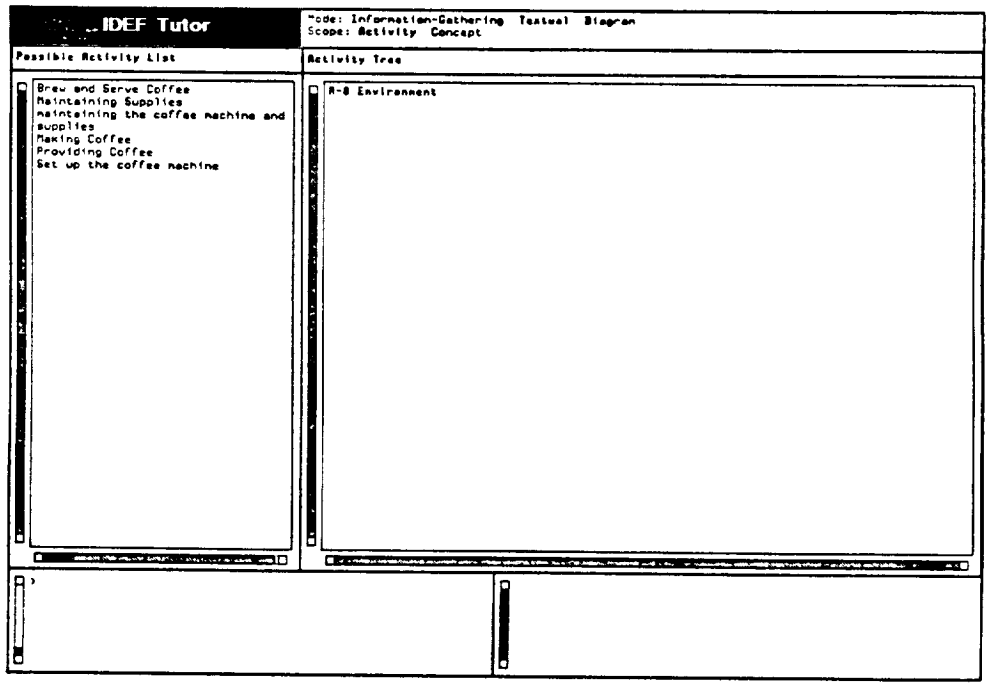


Figure 6.2 Textual-Activity Mode

name to be used in the model and the activity number (A0). Note that the name used in the model does not have to be the verb phrase which names the source data item. The new activity is said to be justified by the source data item. The same source data item cannot be used to justify multiple activities or concepts.

The rest of the activity hierarchy is defined in a similar manner. It may be helpful to fill out at least the "A0" decomposition before adding relationships (sometimes called "flows") to the model. Relationships in IDEF$_0$ define the flow of a concept between activities. Concepts can either be used as inputs, controls, outputs, or mechanisms of activities. Inputs to activities are usually consumed or transformed by the activity, controls determine whether the activity takes place, mechanisms facilitate the activity (but are not consumed or transformed), and outputs are the products of the activity. All activities must have at least one control and one output.

There are two different approaches to adding relationships to a model. The first is to look at each activity and determine that activity's inputs, controls, outputs, and mechanisms. Since the flows being created are only one-sided, we call these "stubs". Once each of the activities is defined, then the stubs of each of the activities are joined to form the relationships between activities. This approach is called "cell modeling".

The second approach is to look at modeling like telling a story. Relationships are created which tell part of the story, like, "The product is manufactured and then sold." Which could be modeled as the "Manufacture Product" activity has an output labeled "product" which is a control on the "Sell Product" activity.

To add the relationships, the first step is to switch to textual-concept mode and define the concepts. Concepts can be identified from the "Possible Concept List" very quickly if the name of the concept is to be the same as the possible concept. If it is, the concept can be created by just clicking left on the possible concept. If a different name is desired, then the "Create Concept" command can be entered from the keyboard followed by the possible concept (select with the mouse) and the name of the concept.

Once the concepts have been defined, switch to diagram mode, still under the "concept" context. The first approach to adding relationships we will describe is the "cell modeling" approach. As such, we will look at the commands used to add stubs to activities. There are four types of stubs: inputs, outputs, controls, and mechanisms. Every activity must have at least one control and one output.

There are two ways to create a stub. The first way is to point at the activity with the mouse, hold the SHIFT key down, and click middle. This will bring up a menu which prompts for the necessary information. First, the type of the stub must be identified by clicking on the appropriate choice. The default selection is "input".

Next, the concept involved is identified. Normally, clicking left on this field, and then clicking right would bring up a menu of possible choices. Unfortunately, under Release 7.2 of Genera, there is a bug which will cause an error if you try to use this procedure. Under Release 7.2, the only way to enter the concept is to type it (use hyphens in place of spaces). Under Release 8.0 you can use the right click selection facility.

Finally, the concept can be tunneled. This means that the concept either appears or disappears from the model without being traced through the hierarchy. For instance, an engineer whom is identified as a mechanism in an activity may be tunneled from the environment since there is no need to see that he or she is a mechanism in the higher level activities. The display of tunneling is rather messy under the current system.

The other method of creating stubs is to use the "Create Stub" command. You will be prompted for the type of the stub, the concept (which can be selected with the mouse from the concept list), and whether to tunnel the concept.

The system will automatically join stubs when appropriate. For instance, if "Prepared Coffee Machine" is the output of one activity and a mechanism on a sibling, the

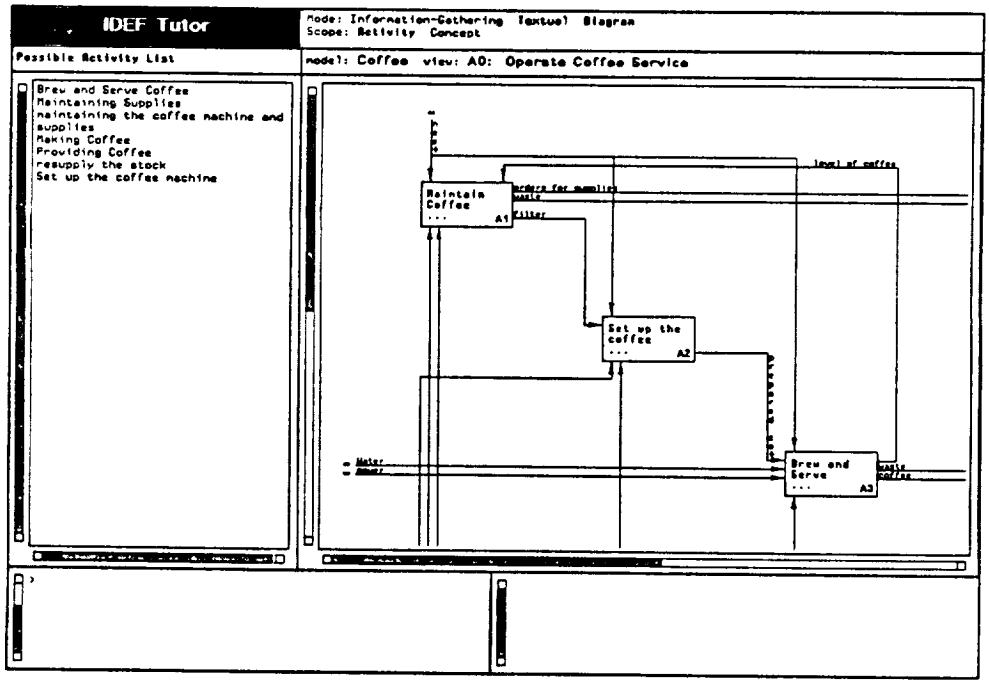


Figure 6.3 Diagram Mode

system will connect the stubs, thus showing the flow from the first to second activities. Consequently, the diagram is valid when all activities have an output and a control and all stubs are either tunneled or connected. Figure 6.3 shows a screen with the completed coffee model shown in diagram mode.

The second method for creating relationships is closer to the "story telling" approach. With this approach, complete relationships are created instead of stubs. To create the relationships you use the "Create Relationship" command. The command will prompt you for the type of relationship (the effect on the destination activity), the source activity, the destination activity, the concept, and the tunneling information. Note that relationships can be created which span many levels in the model. This method does not allow any stubs to be left unconnected.

This completes the general process of creating an IDEF$_0$ model. There are many other commands available for editing the model. These commands are described in [Wells, 88].

6.1.2 Analysis

Now that you have worked through an example scenario, let's look back and see how effective the strategies used in the experimental software turned out to be. To analyze the software, we will refer to the cognitive skills which were identified earlier as being important to IDEF modeling. These skills are: observation, classification, synthesis, decomposition, and language skills.

The software stresses observation and classification skills during the information gathering phase. It is at this point where the modeler must identify the concepts and activities from the source material. The current method the student follows to identify activities and concepts leaves little room for thought. Only possible concepts and activities are mouse sensitive, and little thought is required to differentiate between a noun (concept) from a verb phrase (activity). A true ITS would need to give more freedom to the student.

A true ITS would also need to allow the student to make a wrong choice and then let the student discover later down the road why the choice was wrong. How far down the road is dependent on the issue at hand. The experimental software stops the student immediately upon making a wrong choice. This does not give the student time to necessarily realize why it was a mistake.

Decomposition and synthesis skills are addressed when the student is creating the activity hierarchy and identifying subpart and subtype relationships for concepts. Synthesis skills are also addressed during the diagram building phase.

Finally, language skills are important when creating the model. By comparing the student's model with an expert's model, issues could be identified which cause the

expert's model to communicate the information more clearly than the student's model.

6.2 Context Sensitive Help

No intelligent tutoring system would be complete without context sensitive help. Context sensitive help refers to an on-line service which provides help for specifically requested concepts. In other words, the user of the system does not have to search through pages of documentation for instruction or help on a selected topic. For instance with the IDEF$_0$ tutor the student may wish to see more about "ICOM Codes". It should not be necessary to search through the entire on-line documentation for references to ICOM codes, the system should perform this search for the student. The experimental software shows one approach to providing this service.

6.2.1 Source of the Text

The definitive source for the description of IDEF$_0$ is the Air Force "Yellow Book".[1] This source contains a complete description of the methodology plus the approved definitions for all relevant terminology. Furthermore, this document was used as the source for developing the Concept Dependency Network[2] for IDEF0 training. For these reasons the "Yellow Book" was used as the on-line help document. The text of this document was inserted verbatim, (with the exception of some of the figures) from chapter one through chapter six, page 39.[3] In order to make the document more useful as an on-line IDEF$_0$ User's Manual most of the sections in each of the chapters were broken down into smaller subsections. Each of these subsections (and sub subsections) contain only one or two main topics.

To use the on-line User's Manual the student would select a topic about which he/she requires additional information. The tutor will accept this request and present a list of all section titles that reference the requested topic (see Figure 6.4). For instance, the topic "arrows" will present the user with a list of the titles of all subsections in the user's manual that reference IDEF0 arrows. Selection of one of the titles will

[1] The IDEF$_0$ Yellow Book refers to the "Integrated Computer-Aided Manufacturing (ICAM) Function Modeling Manual (IDEF$_0$)", UM 1102311100, written by SofTech, Inc. for ML/AFWAL Wright-Patterson AFB, OH, 45433 in 1981.

[2] A production quality IDEF$_0$ Tutor, as apposed to the experimental software would have to provide a complete implementation of this concept dependency network.

[3] The insertion of the text stopped at this point because if was felt that the remainder of the document was not necessary for the experimental software.

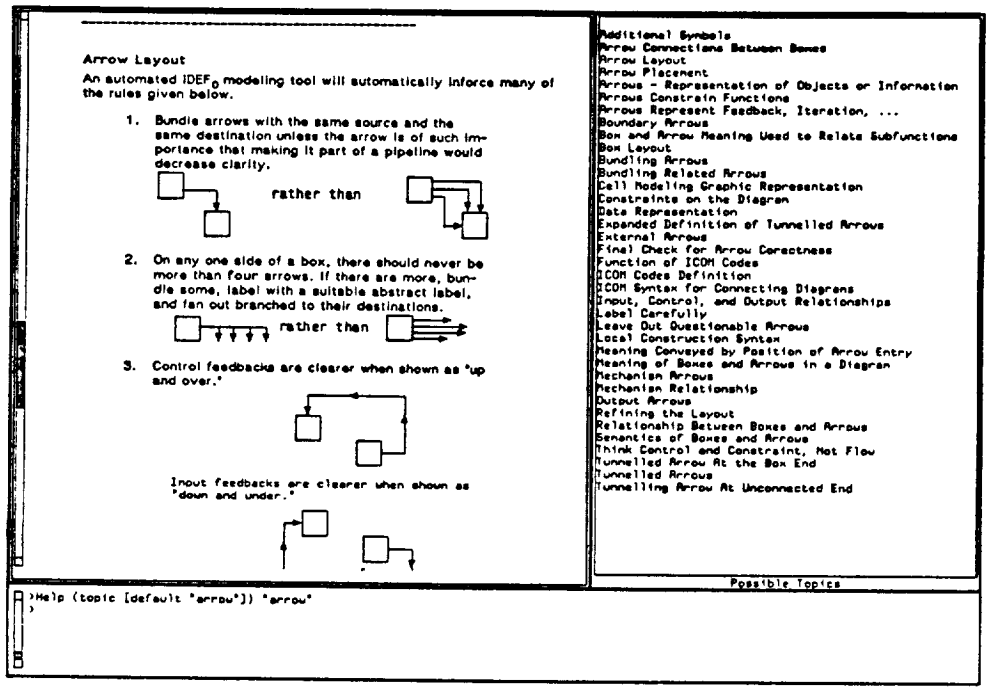


Figure 6.4 Help Screen

provide the user with documentation from the User's Manual that addresses arrows in greater detail.

6.2.2 Evaluation

A direct use of the "IDEF0 Yellow Book" as the IDEF0 Tutorial help manual provides the student with an interpretation of IDEF0 exactly as the creators of the methodology established and described it. Furthermore, since the prototype tutor runs on the Symbolics, the official IDEF0 User's Manual is usable in the Symbolics "Document Examiner" for anyone who simply wishes to browse the manual[4] (familiarity with the Symbolic's Document Examiner is required).

Despite these advantages, the use of the "IDEF0 Yellow Book" would not be the best approach for a production release of an intelligent IDEF0 tutor. An intelligent tutoring system should recognize the specific question about which the student is requesting additional information and provides appropriate answer without any additional refinement by the student. For example, in the "Yellow Book" there are

[4] This can be done by loading the file IDEF0.sab and then selecting the Document Examiner.

several correct answers to the question "What does the term *arrow* mean"? The answer depends upon the context in which the question is asked. Furthermore, even the answers to very specific questions by the student can only be found by browsing and synthesizing large sections of the manual. A production ITS for $IDEF_0$ would require substantial refinement and editing of the contents of the User's Manual to address the needs of the user of an intelligent $IDEF_0$ tutoring system. Thus, an IDEF User's Manual in a production version of an $IDEF_0$ tutor would be based on the "$IDEF_0$ Yellow Book" but the text would very likely be unrecognizable by the author's of the original manual.

7

IDEF Tutor: PC Based Prototype

7.1 Background

While the experimental software was developed in conjunction with the research effort to experiment with cognitive skills for IDEF modeling, the personal computer (PC) based prototype is being developed after the major thrust of the research. The lessons learned in the experimental software have been incorporated into the PC prototype. While the original research entailed developing a general architecture suitable for intelligent IDEF tutoring, the PC prototype is providing a means for further experimentation and testing.

The PC prototype is being done as part of a master's thesis and is not expected to be completed until after January 1991. It is written using Zortech C++ version 2.0. The target platform is an 80386 based personal computer with VGA graphics, a mouse and two or more megabytes of expanded memory.

7.2 Current Status

The tutor architecture relies on a conceptual dependency graph (CDG) to provide a hierarchical course representation. The system must be able to provide good CAI before any intelligence decision making and evaluation may be done. Since it is possible to represent only a limited portion of the IDEF domain with constraints, that portion is one of the focus areas for intelligent decision making for the IDEF tutor. The second focus for intelligent decision making concerns when and what to teach. The third focus area is concerned with teaching the cognitive skills previously identified for IDEF modeling. Most of the instruction will not be truly intelligent as in structured domains such as mathematics.

Graphical User Interface (GUI). The basic tutoring environment is complete. The environment is built on top of Metagraphics's MetaWindow GUI and Ithaca Street Software's Menuet software. The tutor environment consists of controlled entry and identification of user, support for adding courses, support for adding students to the system, windowing and icon support. Scrollable lists, user defined icons, and bitmapped graphics support are common features of the GUI.

Conceptual Dependency Graph and Course Interpreter. The CDG support functions are complete also. A course may now be defined as dependency relationships among lessons. A simple course description language captures the dependency relationships and is read by the course interpreter. In the absence of any intelligent intervention, the tutor automatically builds an agenda of lessons that will represent the course.

Lesson Interpreter. Completion of the CDG and course interpreter paved the way for the lesson interpreter. The course interpreter reads a course file and organizes the lessons. The lesson interpreter is then called to read the tutor language code to conduct each lesson.

Tutor Language. The tutoring language is simple but effective. Currently variables and functions are not supported although they will be added when time permits. Support is provided for a multi-pane windowing look and feel, icons, and primitive branching support.

Student Model. Enough of the system is now built to begin the student modeling module. Progress is expected to be slow due to the difficulty of building this module and the implications for a poor design.

Model Builder Environment (MBE). Based on the lessons gained from the experimental software, components of the model builder environment are being developed. The MBE is specifically being designed to develop the cognitive skills needed for IDEF modeling. These skills are very difficult to develop using the lessons alone.

Test Module. With the capability to develop a course with lessons, testing is rapidly becoming a need. Development of this module will proceed in parallel with that of the student model and MBE.

7.3 Looking Ahead

The goal of the prototype development is to create a functional tutor system in which the architecture and ideas developed may be tested. Currently, the goal is to have a reasonably functional prototype for testing by January 1991. Two environments are being considered as potential test sites:

- Dr. Mayer's class in which the IDEF methods are taught, and
- a class or group in the Texas A&M education department.

The testing should answer many questions including the following:

- can such an architecture effectively support multiple courses?
- can a cognitive skills approach be taken for unstructured environments?
- is intelligent instructional strategy selection by the tutor effective in helping the student learn the material?
- how difficult is it for an instructor to use the system to design courses?
- will instructors actually build on top of the lessons from other courses?
- will lessons from other courses integrate well?

8

Conclusions

8.1 Conclusions and Results

The ideas presented for the development of an ITS system unify many of the ideas that have been explored since the early development of CAI/ICAI/ITS. The key ideas that separate this system are the following:

- identification of specific cognitive skills for IDEF modeling,
- dynamic dependence upon a concept hierarchy for curriculum guidance,
- multiple curriculum depth levels (pedagogical levels),
- integration across concept hierarchies,
- dynamic support of multiple instructional strategies,
- utilization of information concerning the student's background, initial knowledge, and progress to affect curriculum guidance.

An innovative combination of the above components has been described that should provide an ITS system flexible enough for tutoring IDEF modeling and much more.

With the above system, students should no longer be forced to trudge through concepts which they have already mastered just to get to the new concepts. This has financial implications when those students are professionals with limited time for tutoring or briefing sessions. Students should be able to review across domains. No longer should one system be for teaching one domain, another system for another domain, etc. Integration across concept hierarchies should bring many benefits to students and instructors alike.

8.2 Future Directions/Work

Knowledge Acquisition. A knowledge acquisition scheme must be developed to build the model library previously discussed. The difficulty lies in trying to build a library of models in which each model may be done many ways. Different experts may actually produce substantially different models. A method is needed to gain a

consensus on which models are good, poor, and incorrect. A variation of the Delphi technique is being explored.

Tool Support. Perhaps the most obvious enhancement would be to develop a tool set to aid in the creation and modification of the lessons used by the IDEF Tutor system. Such tools would facilitate the development of a lesson including manipulation of bit mapped graphics, animation primitives, and simulation capabilities.

Group Learning. Using networked systems, group learning is possible. A variation of the Delphi technique has potential to facilitate that process. The students could actually participate in the model building process as a group using the Delphi technique. The goal would be for the students to come to a consensus concerning the way to build the model each defending their own view when necessary.

References

Anderson, J.R.; Boyle, C.; and Yost, G. (1985). "The geometry tutor." *Proceedings of Ninth International Joint Conference on Artificial Intelligence*, pp. 1-7. Morgan Kaufmann, Los Altos.

Bloom, B.S., (Editor) (1956). *Taxonomy of Educational Objectives: Handbook I: The Cognitive Domain*. David McKay Co, New York

Brown, J.S., & Burton, R.R. (1978). ???

Brown, J.S.; Burton, R.R.; and DeKleer, J. (1982). "Pedagogical, natural language and knowledge engineering techniques in SOPHIE I,II, and III." In Sleeman, D.H.; and Brown, J.S. (Editors) *Intelligent Tutoring Systems*. Academic Press, London.

Brown, J.S., & VanLehn, K. (1980). "Repair Theory: A generative theory of bugs in procedural skills." *Cognitive Science*, 4, 379-426.

Chomsky, N., (1965). *Aspects of the Theory of Syntax*. MIT Press; Cambridge, Mass.

Clancey, W.J. (1981). "Tutoring rules for guiding a case method dialogue." *International Journal of Man-Machine Studies*, vol. 11, pp, 25-49, (Reprinted in Sleeman, D.H.; and Brown, J.S. (Editors) *Intelligent Tutoring Systems*. Academic Press, London.).

Gagne, R.M. (1962). "The Acquisition of Knowledge." *Psychological Review*, 69, 355-365.

Genesereth, M.R. (1982). "The role of plans in intelligent teaching systems." In Sleeman, D.H.; and Brown, J.S. (Editors) *Intelligent Tutoring Systems*. Academic Press, London.

Langley, P.; and Ohlsson, S. (1984). "Automated cognitive modeling." *Proceedings of American Association of Artificial Intelligence*, pp. 193-197, Morgan Kaufman, Los Altos.

Lee, M.C. (Winter 1988) "An Expert System Construction Approach to CAI Development," *Journal of Research on Computing in Education*.

Lesgold, Alan. (1988) "Toward a Theory of Curriculum for Use in Designing Intelligent Instructional Systems." *Learning Issues for Intelligent Tutoring Systems*. pp. 114-137, Springer-Verlag, New York.

Mayer, R. J. (Editor)(1990). *IDEF0 Function Modeling: A Reconstruction of the Original Air Force Wright Aeronautical Laboratory Techical Report AFWAL-TR-81-4023 (The IDEF0 Yellow Book).* Knowledge Based Systems, Inc., College Station, Texas.

Sleeman, D.H.; and Brown, J.S., (Editors)(1982) *Intelligent Tutoring Systems.* Academic Press, London.

Soloway, Elliot and VanLehn, Kurt. (1987) *Proceeding of AAAI-87: AI and Education.*

Towne, D.M., and Munro, Allen. (1987) "The Intelligent Maintenance Training System." *Intelligent Tutoring Systems: Lessons Learned* (edited by Psotka, Massey & Mutter) pp 479-530.

Uhr. L. (1969). "Teaching machine programs that generate problems as a function of interaction with students." *Proceedings of the 24th National ACM Conference*, pp 125-134. Association for Computing Machinery, New York.

VanLehn, Kurt, (1986) "Student Modeling," *Foundations of Intelligent Tutoring Systems* (edited by Polson & Richardson), pp. 55-78, Lawrence Erlbaum.

APPENDIX A

IDEF$_0$ CONCEPTS REFERENCE

The following represents the concepts isolated for IDEF$_0$. These concepts were used to develop the Concept Dependency Graph (CDG) for IDEF$_0$. The concepts listed in this reference were extracted from a reconstructed version of the original Air Force Technical Report AFWAL-TR-4023 [IDEF$_0$90]. Each concept has been assigned a number for identification in the CDG. Each numbered concept is represented by a node in the CDG. The identifier enclosed in brackets refers to the location on the originally developed "wall chart" version of the CDG.

100 Overview Slideshow of Expert [4A]

101 Definition of "IDEF" [3A]

102 Definition of "information model" [3A]

103 Definition of "enterprise" [3A]

104 Definition of "function model" & "dynamics model" [3A]

105 Relationship between architecture and method [3A]

106 Definition of a "modeling team" [3A]

107 Definition of a "project manager" [3A]

108 Definition of "sources" [3A]

109 Definition of "viewpoint" [3A]

- Determines what can be "seen" within the context and from what "slant"
- States the author's position as an observer of or participant in the system for the benefit of the audience

110 Definition of "context setting" [3A]

- Established the subject of the model as part of a larger whole
- Creates a boundary with the environment by describing external interfaces

200 Background [3B]

- IDEF (ICAM definition)
- IDEF methods ($IDEF_0$,$IDEF_1$,$IDEF_2$)

201 Purpose of $IDEF_0$ [2A]

202 IDEF models - architectures [2A]

203 Method (IDEF), Means (architecture), and End (improving mfg prod) [2A]

204 $IDEF_0$'s relationship to SADT [2A]

205 Cell Modeling [2A]

206 Six Basic $IDEF_0$ Concepts [2A]

- Cell modeling graphic representation
- Conciseness
- Communication
- Rigor and precision
- Methodology
- Organization

300 Basic $IDEF_0$ Concepts [2B]

- Purpose of $IDEF_0$
- IDEF models - architectures
- method (IDEF), means (architecture), and end (improving mfg productivity)
- $IDEF_0$'s relationship to SADT
- Cell Modeling
- Six Basic $IDEF_0$ Concepts

400 Roles of the Modeling Group Members [4C]

- Project manager
- Modeler
- Sources
- Experts
- Review committee

401 Model Definition[1D]

- Decomposition
- Decomposition Rules
- FEO's
- Representation of a system; may describe what a system is, what it does, and what things it works on
- Composed of: diagrams + FEO + text + glossary (all are cross-referenced)
- Series of diagrams with supportive documentation that break a complex subject into its component parts
- A node index or table of contents must be provided
- Final model represents the agreement of the author and reviewers on a representation of the system being modeled from a given viewpoint and for a given purpose
- Model orientation (includes context, viewpoint, and purpose

401-1 Decomposition [1C]

401-1 Decomposition Rules [1C]

- Model decomposed into 3-6 submodules; background reason
- Relationships between modules captured by arrows
- Every submodule contains only those elements within the scope of its parent module

401-3 FEO [1C]

- "For Exposition Only"

- May contain more than six boxes, partial arrow structures, or anything needed by the author to illustrate a point
- Node numbers contain "F" (e.g. A2F)

402 IDEF0 Symbols [1D]

- Diagrams
- Boxes
- Arrows
- Box/Arrow Relationship

402-1 Diagrams [1C]

- Composed of boxes and arrow
- Composed of 3-6 boxes
- A one-box diagram is provided as the "context" or parent of an entire model
- By convention, the diagram has the node number "A-0" (A minus zero)
- Main path of a diagram
- Diagram interpretation

402-2 Boxes [1C]

- Represent functions
- Numbered in its lower right corner
- Bottom reserved to indicate a mechanism; i.e. the person or device which carries out the function
- Represent collections of related functions, not just monolithic actions
- May perform various parts of its function under different circumstances, using different combinations of its input and controls and producing different outputs

402-21 Functions [1B]

- Represented by boxes
- Include activities, actions, processes, or operations
- Described by an active very phrase written inside the box
- Anything that can be named with an active verb phrase

- Examples: tighten, attach, measure, assemble, transcribe, evaluate, classify, construct, solve, adapt
- Not expressed as nouns
- Transform data (from left to right)

402-22 Mechanisms [1B]

- Arrows generally point upward; toward the bottom of the box
- Shows how that function is accomplished
- Diagrams drawn without mechanism show what functions a system must perform
- Downward pointing mechanism arrow ("call") indicates a "system" that completely performs the function of the box
- May be the output of other boxes

402-3 Arrows [1C]

- Represent objects or information needed by or produced by the function
- Labeled with a noun phrase written beside the arrow
- Act as constraints that define the boxes, not sequences or flows of functions
- Roles: input (left), control (top), output
- Easy to show feedback, iteration, continuous processes, and overlapping
- Types: internal, boundary

402-31 Internal Arrow [1B]

- Both ends connected to boxes shown on the diagram

402-32 Boundary Arrow [1B]

- One end unconnected, implying production by or use by a function outside the scope of the diagram
- Unconnected at one end represent data that is supplied or consumed outside the scope of the diagram; the source or destination of these boundary arrows can only be found by examining the parent diagram

402-4 Box/Arrow Relationship [1C]

- Arrows are constraints that define the boxes, not sequences or flows or functions

- Side of box which arrows enters or leaves represents the arrow's role

402-41 Arrow Connections Between Boxes [1B]

402-42 Mechanism Arrows [1B]

403 Additional IDEF0 Symbols [1D]

- Reference Expressions
- Continuing Arrows Across Diagram Boundaries
- Coding Boundary Arrows
- Tunnelling
- Decomposition Example

403-1 Tunnelling [2C]

- Tunnelled arrows *indicate* that the data conveyed by these arrows was not relevant to a particular level of detail
- Tunnelling an arrow *where it connects to a box* indicates that the data conveyed is not necessary at the next level of decomposition
- Tunnelling an arrow *at the unconnected end* indicates that the data conveyed is not relevant to or supplied by the parent diagram
- *Parenthesizing the unconnected ends* says "this arrow does not appear in the parent diagram. It has no ICOM code."
- *Parenthesizing the end where the arrow connects to the box* says "this arrow does not appear in detail diagrams. Its ICOM code is not tracked from here on and may never be explicitly referenced"
- Note: it is possible for an arrow to have a parenthisized arrowhead, disappear for one or more levels of detail, and then be reintroduced at some specific level of detail with a parenthesized end

403-11 ICOM Codes [1C]

- Notation used to specify the matching connections
- Letter "I" (input), "C" (control), "O" (output), or "M" (mechanism) written near the unconnected end of each boundary arrow on the detail diagram to identify the arrow's role in the parent box
- Letter followed by a number giving the position at which the arrow is shown entering or leaving the parent box, numbering left to right, top to bottom

- Must be written at the unconnected ends of all boundary arrows except for the very topmost diagram in a model and on tunneled arrows

403-2 Reference Expressions [2C]

- Node Numbers
- Model Names (and Node Numbers)

403-21 Node Names [2C]

- Used to indicate the position of any diagram or box in the hierarchy
- Begin with the letter "A" to identify them as "activity" or function diagrams
- By convention, the diagram has the node number "A-0" (A minus zero)
- FEO node numbers contain "F" (e.g. A2F)
- Used to indicate the decomposition of a box in a diagram
- If a box has been decomposed, the node number of the diagram which represents the decomposition is written outside the box under the right hand corner

403-22 Model Names [2C]

- Each model must have a name for identification (e.g. TOPIC)
- Diagrams in the model are referred to by adding a slash and the node number to the name (e.g. TOPIC/A3)

403-3 Continuing Arrows Across Diagram Boundaries [2C]

403-4 Coding Boundary Arrows [2C]

403-5 Decomposition Example [2C]

404 Time/Sequence [2D]

- Not explicit in IDEF$_0$ diagrams

500 Understanding IDEF$_0$ Diagrams [2D]

- Model Definition

- $IDEF_0$ Symbols
- Additional $IDEF_0$ Symbols
- Time/Sequence

501 IDEF Teamwork Discipline [3E]

- Authors
- Reviewers
- Commenters
- Readers
- Viewpoint/Purpose

501-1 Authors [3D]

501-2 Reviewers [3D]

501-3 Commenters [3D]

501-4 Readers [3D]

501-5 Viewpoint/Purpose [3D]

502 IDEF Kit Cycle [3E]

- Personnel Roles
- Guidelines for Authors and Commentors
- Author/Reader Cycle

502-1 Personnel Roles [3D]

- Authors
- Commenter

502-2 Guidelines for Authors and Commenters [3D]

- Commenter Guidelines
- Author/Commenter Interchanges
- Meeting Rules

502-21 Commenter Guidelines [3D]

502-22 Meeting Rules [3D]

- Until comments and reactions are on paper, commenters and authors are discouraged from conversing
- When a meeting is required:
    - each meeting should be limited in length
    - each session must start with a specific agenda of topics to be considered and must stick to these topics
    - each session should terminated when the participants agree that the level of productivity has dropped and individual efforts would be more rewarding
    - Each session must end with an agreed list of action items which may include the scheduling of follow-up sessions with specified agendas
    - In each session, a "scribe" should be designated to take minutes and note actions, decisions, and topics
    - Serious unresolved differences should be handled professionally, my documenting both sides of the picture
- Result of the meeting should be written resolution of the issues or a list of issues to be settled by appropriate managerial decision

502-23 Author/Commenter Interchanges [3D]

502-3 Author/Reader Cycle [3D]

- model author — kit1 — expert reviewer
- expert reviewer — kit2 — model author
- model author — kit3 — expert reviewer
- kit1 = original author's model
- kit2 = kit1 with reviewer's comments
- kit3 = kit2 with author's response

503 IDEF Kits [4E]

- Technical Document
- Cover Sheet
- Preparing a Standard Kit

- IDEF Kit Types

503-1 Cover Sheet for a Standard Kit [4D]

503-2 Preparing a Standard Kit [4D]

504-1 Working Information [4D]

504-2 Number Field [4D]

504-4 Title Field [4D]

504-4 Message Field [4D]

504 Standard Diagram Form [5E]

- Working Information
- Message Field
- Title Field
- Number Field

505 Maintaining Files [6E]

- Standard Kit File
- Summary Kit File
- Working File

506 IDEF Model Walk-Through Procedure [6E]

- Scan the Diagram
- Look at the Parent
- Connect Parent Box and the Detailed Diagram
- Examine Internal Arrow Pattern
- Read Supportive Documentation
- Set the Status of the Diagram

600 IDEF Kit Cycle, Forms, and Procedures [5E]

- IDEF Teamwork Discipline
- IDEF Kit Cycle

- IDEF Kits
- Standard Diagram Form
- Maintaining Files
- IDEF Model Walk-Through Procedure

601 Approaching a Model [1F]

602 Miscellaneous for Reading IDEF$_0$ Diagrams [1F]

- Node Index
- Page-Pair Format
- Read Top Down

603 Diagram Reading Steps [1F]

- Scan the boxes of the diagram to gain an impression of what is being described.
- Refer back to the parent diagram and note the arrow connections to the diagram. Try to identify a "most important" input, control, and an output.
- Consider the arrows of the current diagram. Try to determine if there is a main path linking the "most important" input or control and the "most important" output
- Mentally walk through the diagram, from upper left to lower right, using the main path as a guide. Note how other arrows interact with each box.
- Determine if there are secondary paths. Check the story being told by the diagram by considering how familiar situations are handled.
- Check to see if a related "FEO" diagram exists.
- Finally, read the text and glossary if provided.

603-1 Only that which is explicitly stated is necessarily implied [2F]

603-2 Constraint Diagram [2F]

603-3 Constraints Omit How and When [2F]

603-4 Multiple Inputs, Control, and Outputs [2F]

- multiple inputs, controls, and outputs are allowed for each box

- in general: cannot assume that any output can be produced without all entries present or that any output requires all entries for its production
- in general: some form of further detailing will specify the exact relationship of inputs and control to outputs

604 Semantics of Boxes and Arrows [2F]

- Only that which is explicitly stated is necessarily implied
- Constraint Diagrams
- Constraints Omit How and When
- Multiple Inputs, Controls, and Outputs

604-1 Node Index [1F]

- Structure of functions and subfunctions formatted in an index format
- Similar to the format of a table of contents and the format of an "indentured parts list" (i.e. bill of materials) used in manufacturing and engineering
- See "node index order"

604-2 Node Index Order [1F]

- All detail drawings relating to one box on a diagram are presented before the details of the next box
- Used so that related diagrams will be grouped together in the same order used in the table of contents

604-3 Page-Pair Format [1F]

- Each diagram and the entire text associated with it appear on a pair of facing pages

604-4 Read Top Down [1F]

700 Reading IDEF$_0$ Diagrams [2G]

- Approaching a Model
- Diagram Reading Steps
- Semantics of Boxes and Arrows
- Miscellaneous Notes for Reading IDEF$_0$ Diagrams
- Reading is done top-down

- If specific details about a model are needed, the node index is used to descend through the levels to the required detail
- Published model is bound in "page-pair" format and "node index" order

701 Interview Process [4F]

- Fact Finding
- Problem Identification
- Solution Discussion
- IDEF Author/Commenter Talk Session

702 Interview Kit [4F]

- Cover Page (kit cover)
- Interview and Record Follow-up
- Activity and Data List
- Interview Agenda
- Interview Notes and Rough Diagrams

703 Introduction to Data Collection for IDEF Modeling [4F]

704 Interview Guidelines [5G]

- Interview Preparation
- Interview Initialization
- Conducting the Interview
- Termination
- Finalization

800 Data Collecting for IDEF Modeling [5G]

- Introduction to Data Collecting for IDEF Modeling
- Interview Process
- Interview Kit
- Interview Guidelines

801 Basic Steps for Authoring $IDEF_0$ Diagrams [2I]

- Selecting A Context, Viewpoint and Purpose
- Creating the Context Diagram
- Creating the Top Most Diagram
- Creating Subsequent Diagrams
- Creating Supporting Material
- Selecting a Box to Decompose
- Author Activities
- Bound the subject matter more precisely than the title of the function box suggests. This is done with a list of data (objects or information) acted on or processed by the function
- Study the bounded set of subject matter and form possible subfunctions of the total function
- Look for natural patterns of connections of those subfunctions
- Split and combine subfunctions to make other boxes
- Draw a final version of the diagram with careful attention to layout and clarity

801-1 Creating Subsequent Diagrams [2I]

801-2 Creating the Context Diagram [2H]

801-3 Selecting a Context, Viewpoint and Purpose [2H]

801-4 Author Activities [2H]

- Data Gathering Phase
- Structuring Phase
- Presentation Phase
- Interaction Phase

801-41 Data Gathering Phase [2G]

801-42 Structuring Phase [2G]

801-43 Presentation Phase [2G]

801-44 Interaction Phase [2G]

801-5 Selecting a Box to Decompose [2H]

801-6 Creating the Top Most Diagram [2H]

801-7Creating Supporting Material [2I]

802 Model Creation Requirements [3I]

- Its purpose and viewpoint must match the stated purpose and viewpoint of the overall model
- Its boundary arrows must correspond to those of its parent diagram
- Its content must be exactly everything in its parent box

803 Drawing An IDEF$_0$ Diagram [3I]

- Generating Function Boxes
- Creating Interface Arrows
- Level of Effort
- Note: most subjective and creative activity of the modeling process
- Create a relevant, but not yet structured list of data. List items within the context of the parent box that first comes to mind. Group items to show similarities.
- Name functions that act on the listed data and draw boxes around the names
- Sketch appropriate arrows. As each box is drawn, leave arrow stubs to make the box more meaningful. Make complete connections as it becomes obvious what the diagram is saying.
- Draft a layout that presents the clearest box and arrow arrangement. Bundle arrows together if the structure is too detailed. Leave only the essential elements, and modify diagram as necessary.
- Create text, glossary, and FEO diagrams, if necessary, to highlight aspects which are important. Propose changes, if needed, in the parent diagram.

803-1 Generating Function Boxes [3G]

- Make function box names verb phrases
- In most cases, layout boxes diagonally from upper left to lower right. While any layout which makes clear the author's intent is acceptable, vertical or horizontal formats tend to crowd arrows and hinder good structured analysis style.

- Boxes placed in the upper left "dominate" boxes placed lower and to the right through the control arrows that link them. This standard style makes it easier for readers to understand your meaning.
- Number each box in its lower right corner. Assign the box numbers from left to right and from top to bottom. The leading digits of the box's complete node number are the same as this diagram's node number.
- On working or draft copies, write the author C-number below the lower right corner of any box that is decomposed.
- No diagram may contain more than six boxes.

803-2 Creating Interface Arrows [3G]

- Think control and constraint, not flow
- Avoid cluttering diagrams with too much information and too many arrows
- Leave out questionable arrows

803-3 Level of Effort [3G]

- Reworking of diagrams will always be a necessary part of the process
- Use a review cycle to make progress on paper

804 Graphic Layout [3I]

- Constraints on the Diagram
- Arrow Placement
- Arrow Layout

804-1 Constraints on the Diagram [3I]

804-2 Arrow Placement [3I]

804-3 Arrow Layout [3H]

805 Redrawing An IDEF$_0$ Diagram [3H]

- Modifying Boxes
- Bundling Arrows
- Proposing Modifications to the Context
- ICOM Syntax for Connecting Diagrams

- Global Construction Syntax
- Model Construction Syntax

807-21 Local Construction Syntax [4G]

807-22 Global Construction Syntax [4G]

807-23 Model Construction Syntax [4G]

807-3 Measures and Types of Cohesion [4H]

- Logical
- Temporal
- Procedural
- Communicational
- Sequential
- Functional

807-4 Metrics Based on Coupling and Cohesion [4H]

- Relation to Other Systems Engineering Properties
- Measures and Types of Coupling

807-41 Measures and Types of Coupling [4G]

- Viewpoint of the Description
- Nature of Connections (normal, control, pathological
- Structure of the Connection (environmental, record, abstract)

807-41-1 Viewpoint of the Description [4G]

807-41-2 Structure of Connections [4G]

807-41-3 Nature of Description [4G]

807-42 Relation to other Systems Engineering Properties [4G]

807-5 Assessing Coupling/Cohesion in IDEF [4H]

807-6 Relationship between Coupling/Cohesion [4I]

805-1 Modifying Boxes [3G]

805-2 Bundling Arrows [3G]

805-3 Proposing Modifications to the Context [3G]

805-4 ICOM Syntax for Connecting Diagrams [3G]

806 Writing Text [3I]

- References and Notes
- Writing the A-0 Text

806-1 References and Notes [3I]

806-2 Writing the A-0 Text [3I]

807 Model Quality Checklist [4I]

- Syntax
- Semantics
- Relationship Between Coupling/Cohesion
- Metrics Based on Coupling and Cohesion
- Measures and Types of Cohesion
- Assessing Coupling/Cohesion in IDEF

807-1 Semantics [4I]

- Completeness
- Conciseness
- Consistency
- Correctness
- Complexity/Understandability

807-2 Syntax [4H]

- Local Construction Syntax

900 Author's Guide To Creating IDEF$_0$ Diagrams [3J]

- Model Creation Requirements (3)
- Basic Steps for Authoring IDEF$_0$ Diagrams (5)
- Drawings an IDEF$_0$ Diagram
- Redrawing an IDEF$_0$ Diagram
- Graphic Layout
- Writing Text
- Model Quality Checklist

1000 IDEF$_0$ [3J]

APPENDIX B

$IDEF_0$ CONCEPT DEPENDENCY GRAPH

The following diagrams represent the Concept Dependency Graph (CDG) developed for $IDEF_0$. The first figure provides a view of the top level major nodes. All figures after the first figure show the composition of one of the major nodes in the IDEF0 CDG or the composition of a subnode within one of the major nodes.

The CDG was developed using material from a reconstructed version of the original Air Force Technical Report AFWAL-TR-4023 [$IDEF_0$90].

Each figure is read left to right. Nodes on the left must preceed nodes on the right.

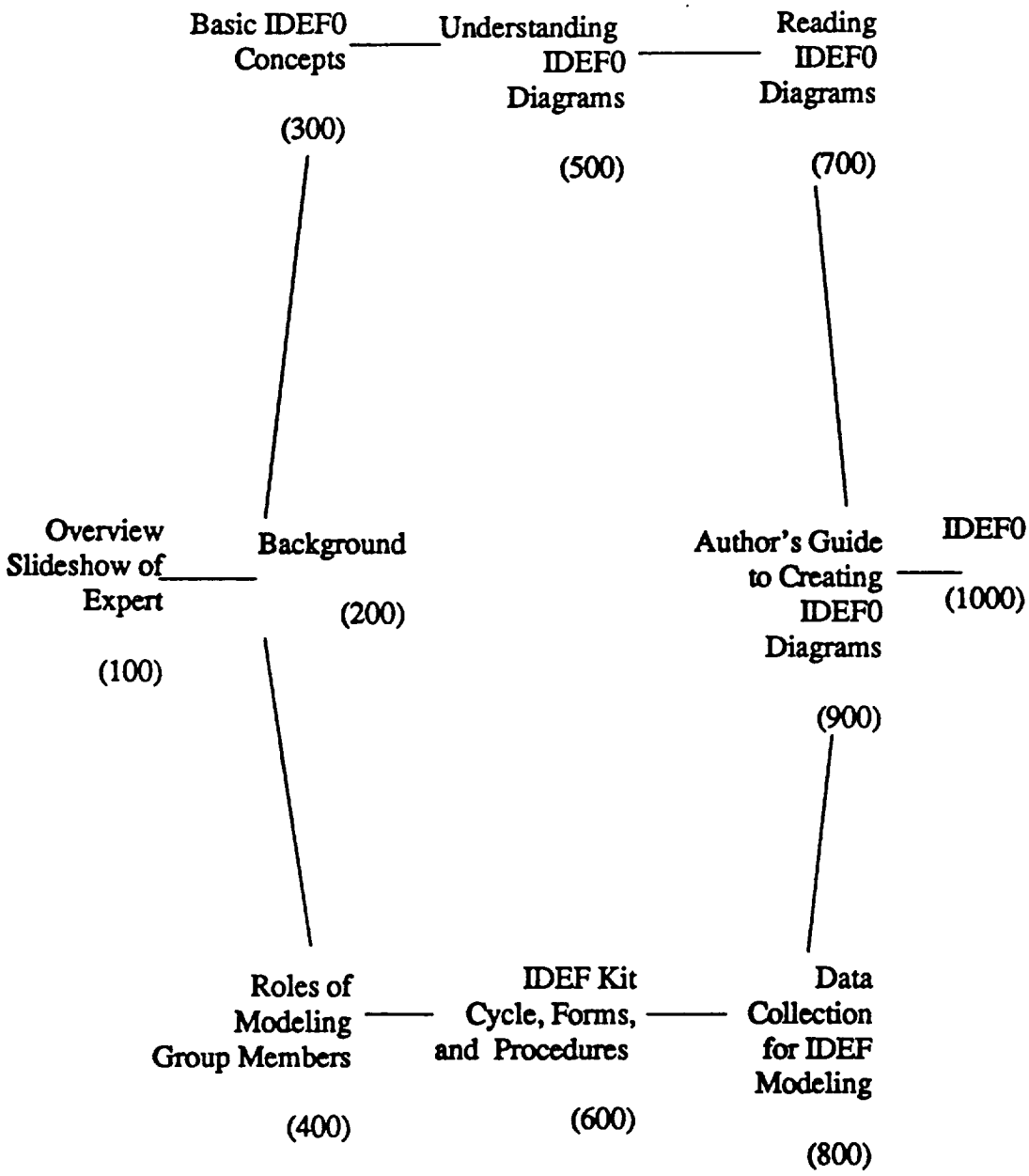


Figure B.1 IDEF0 CDG Top Level Overview

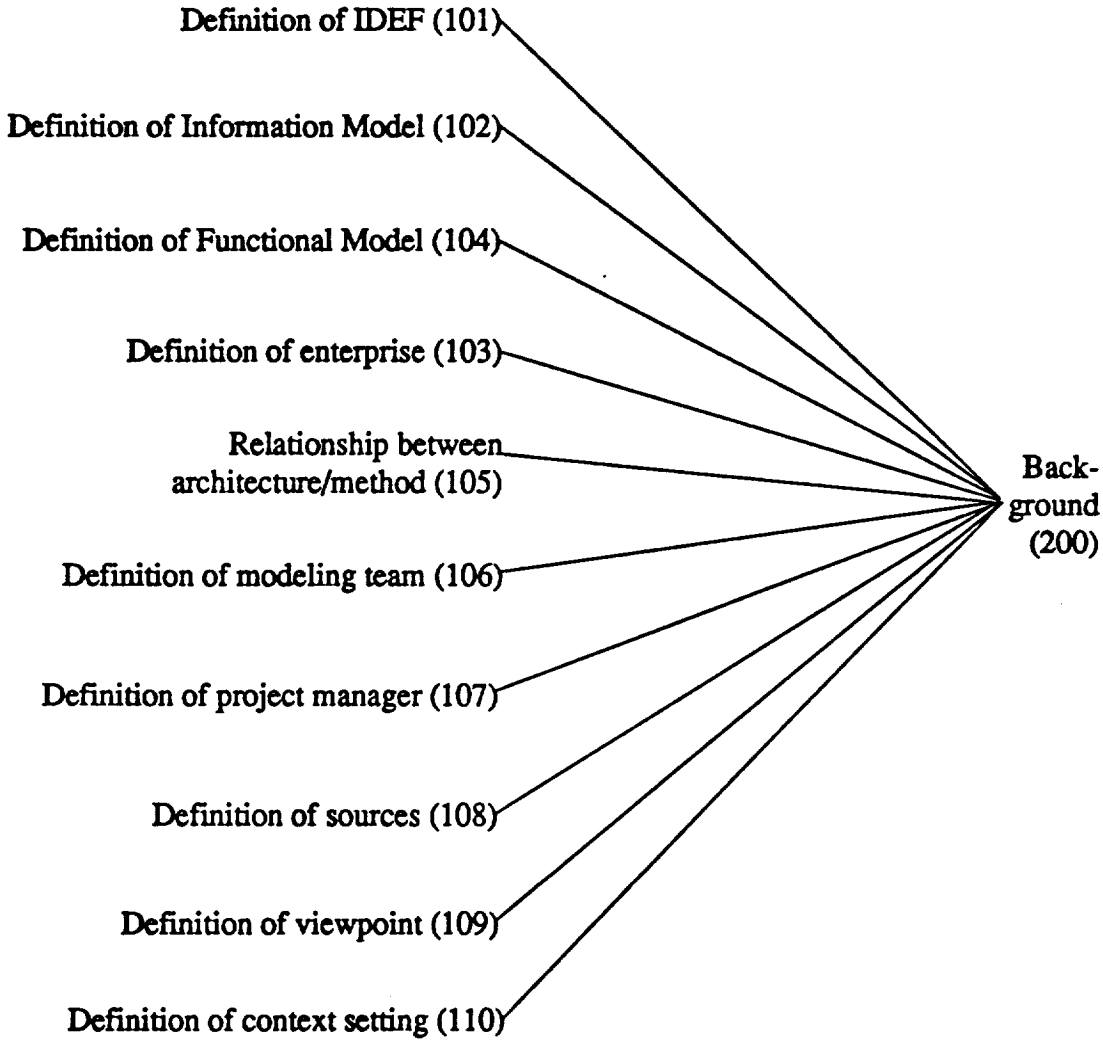


Figure B.2 IDEF0 CDG Background Node 200

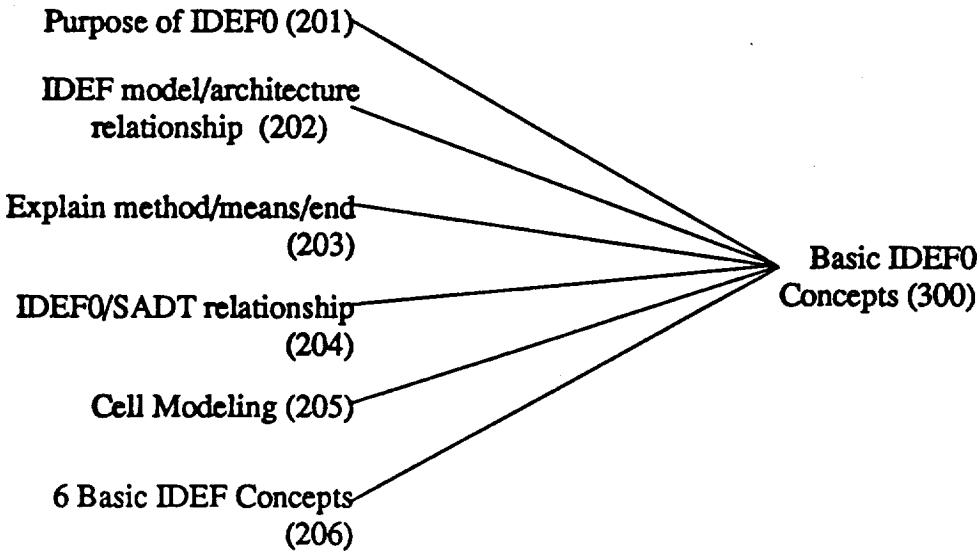


Figure B.3 IDEF0 CDG Basic IDEF0 Concepts 300

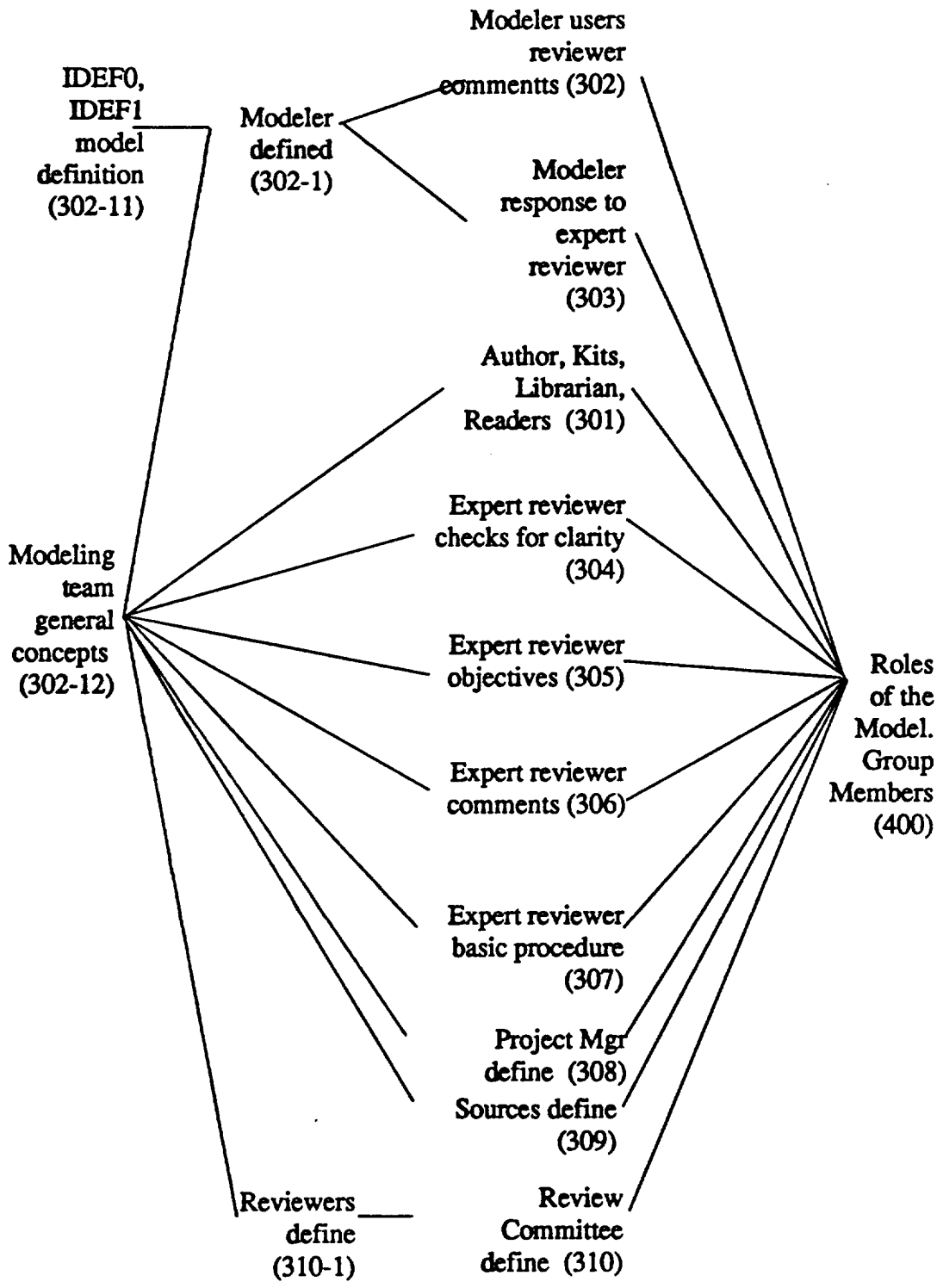


Figure B.4 IDEF0 CDG Modeling Group Roles 400

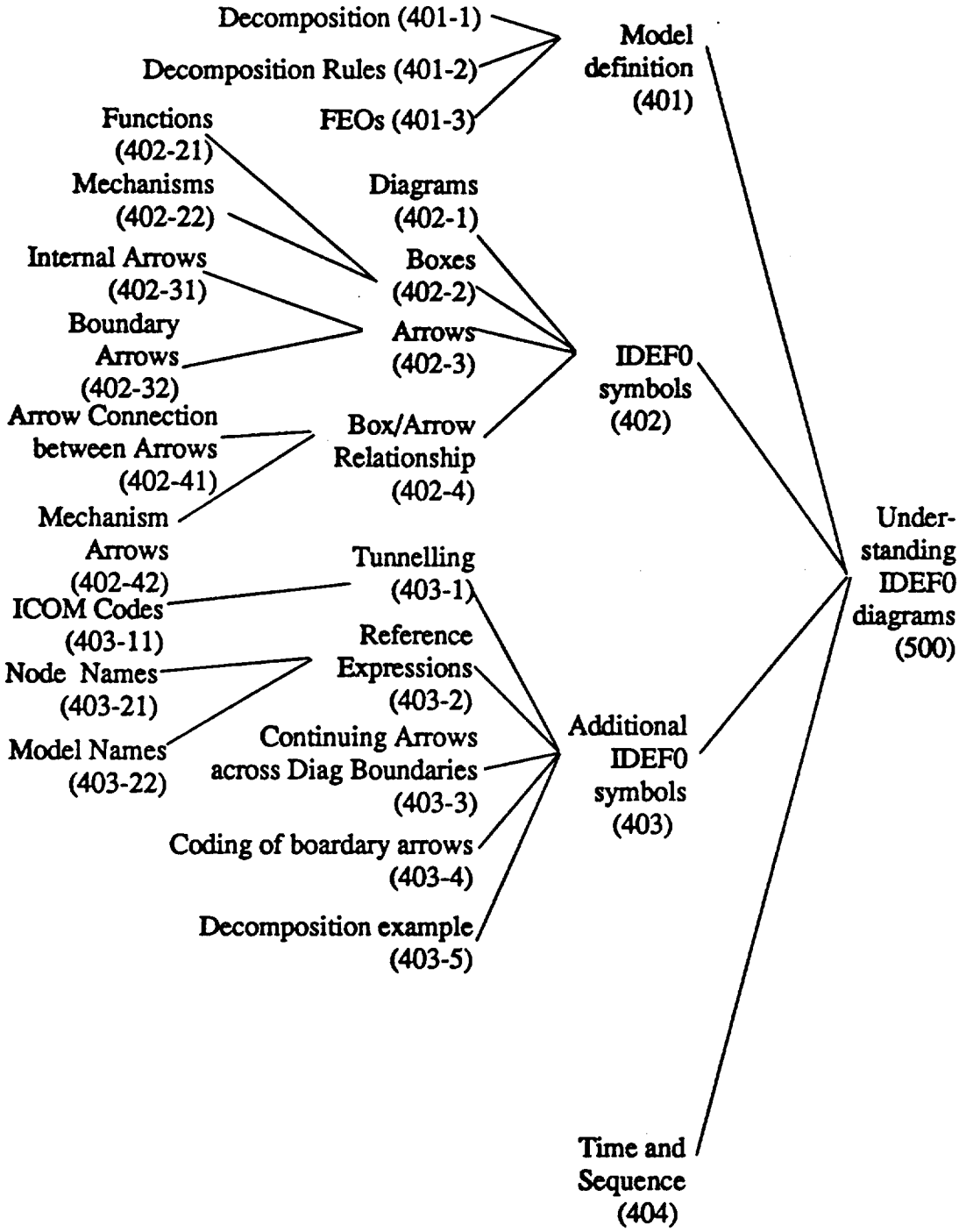


Figure B.5 IDEF0 CDG Understanding Diagrams 500

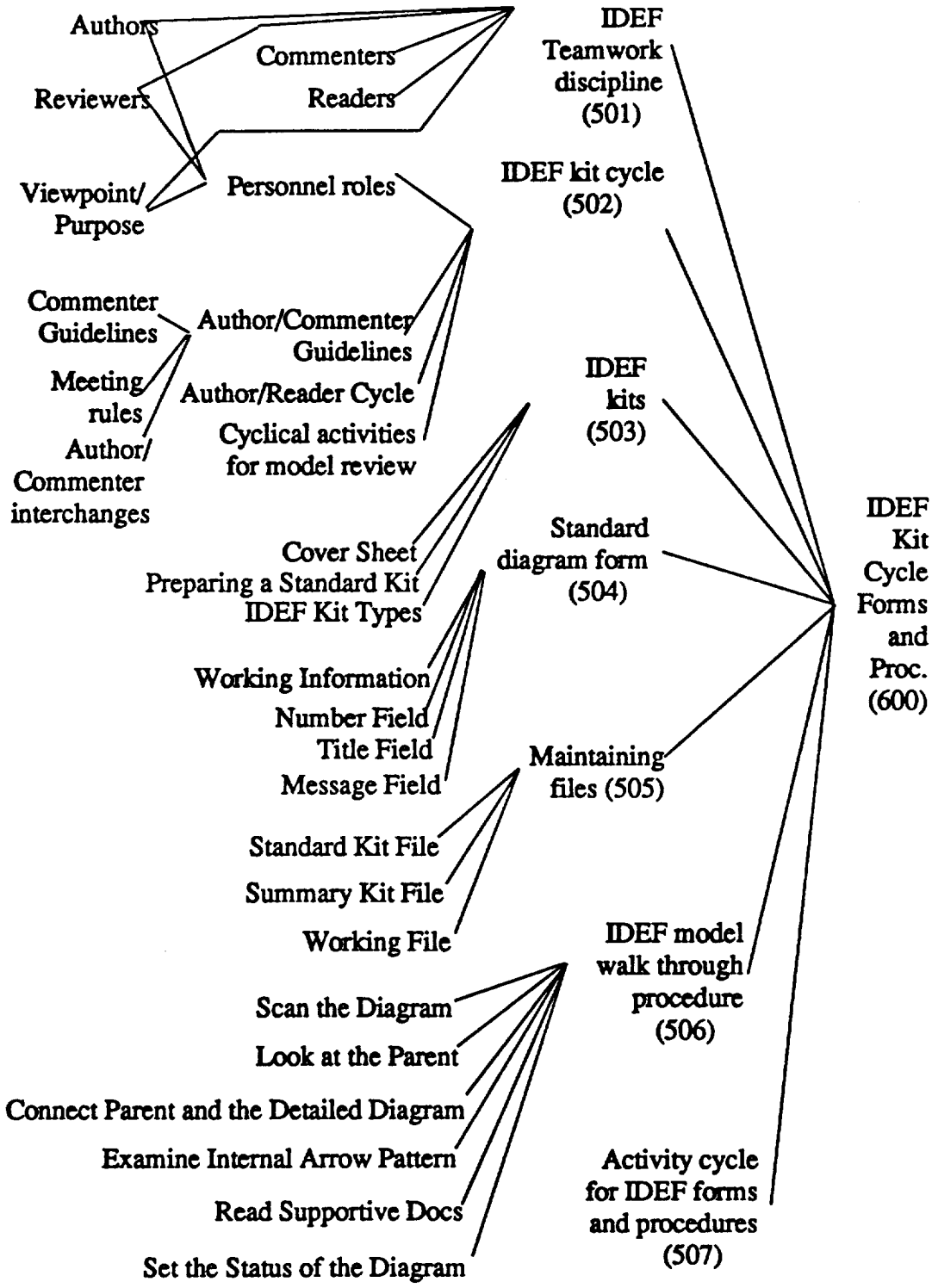


Figure B.6 IDEF0 CDG Kit Cycle, Forms, Procs 600

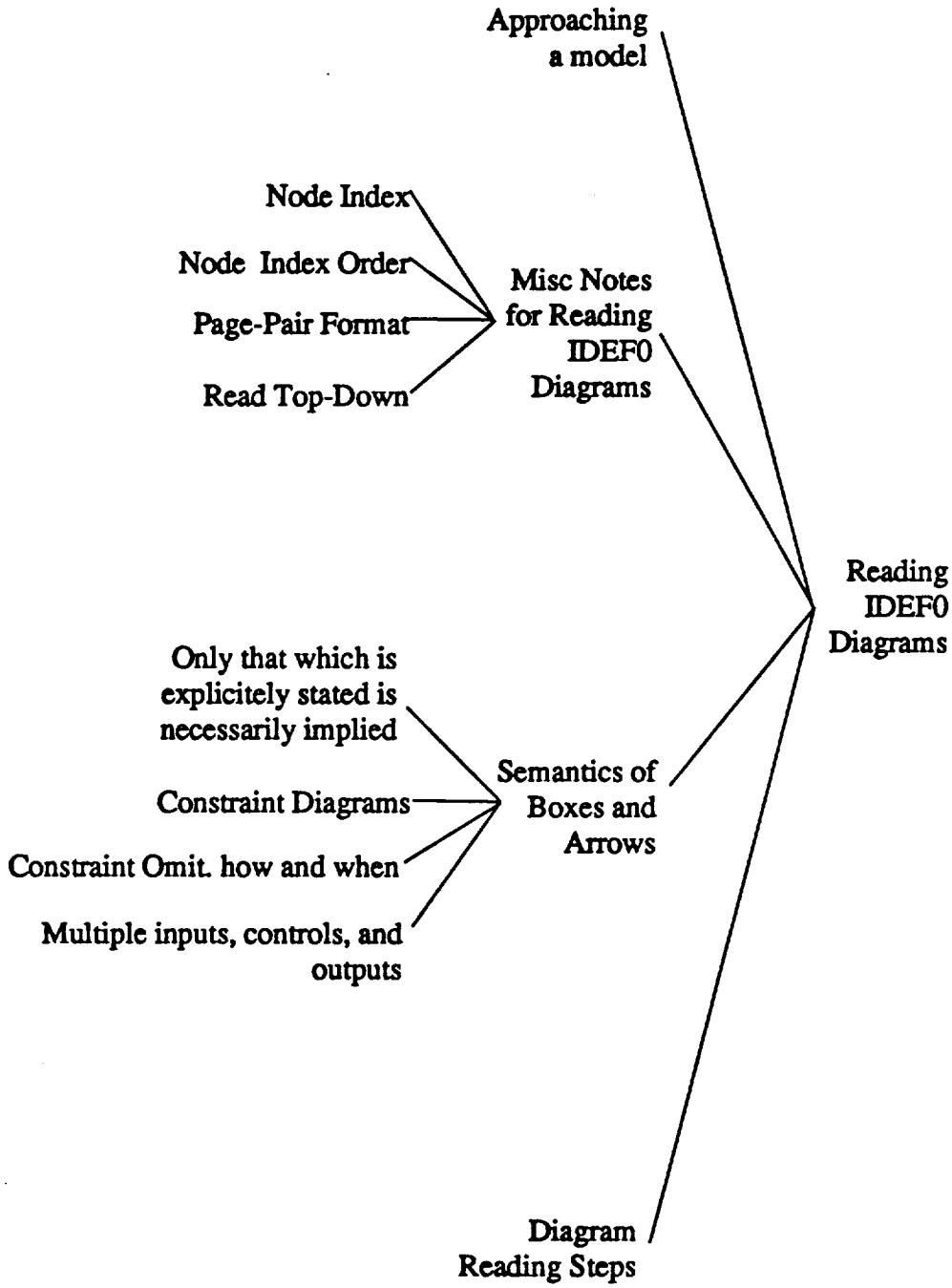


Figure B.7 IDEF0 CDG Reading Diagrams 700

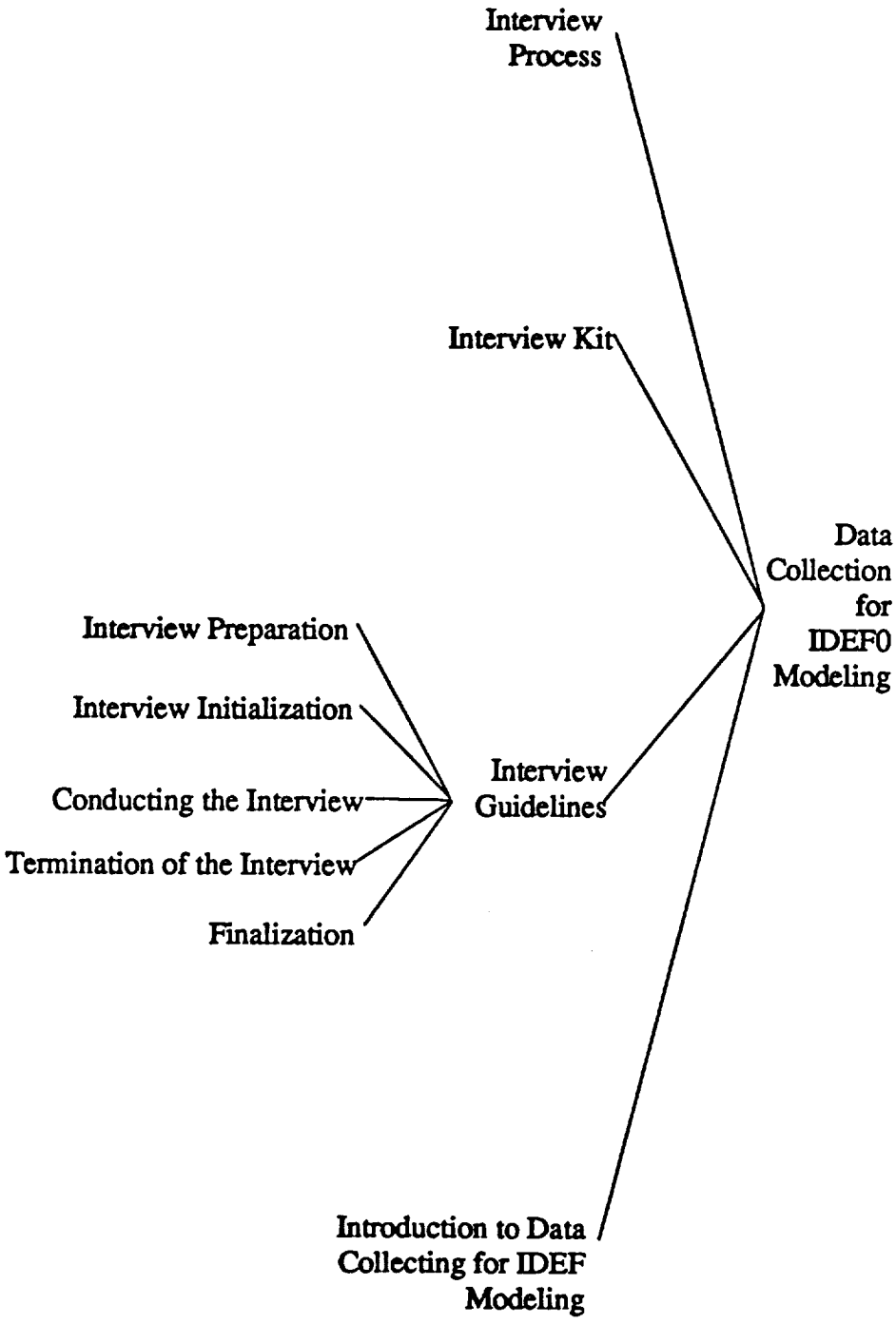


Figure B.8 IDEF0 CDG Data Collection 800

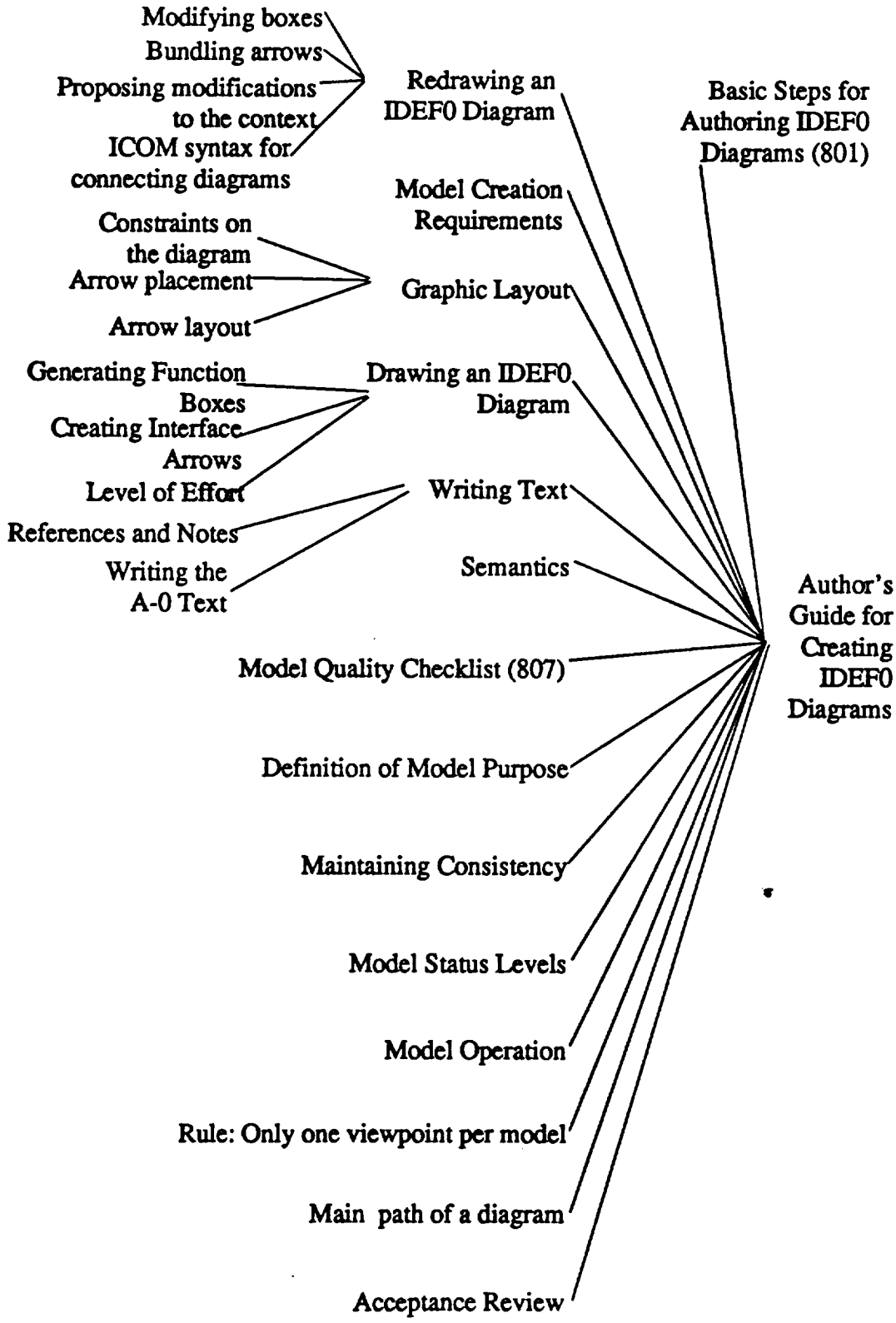


Figure B.9 IDEF0 CDG Author's Guide 900

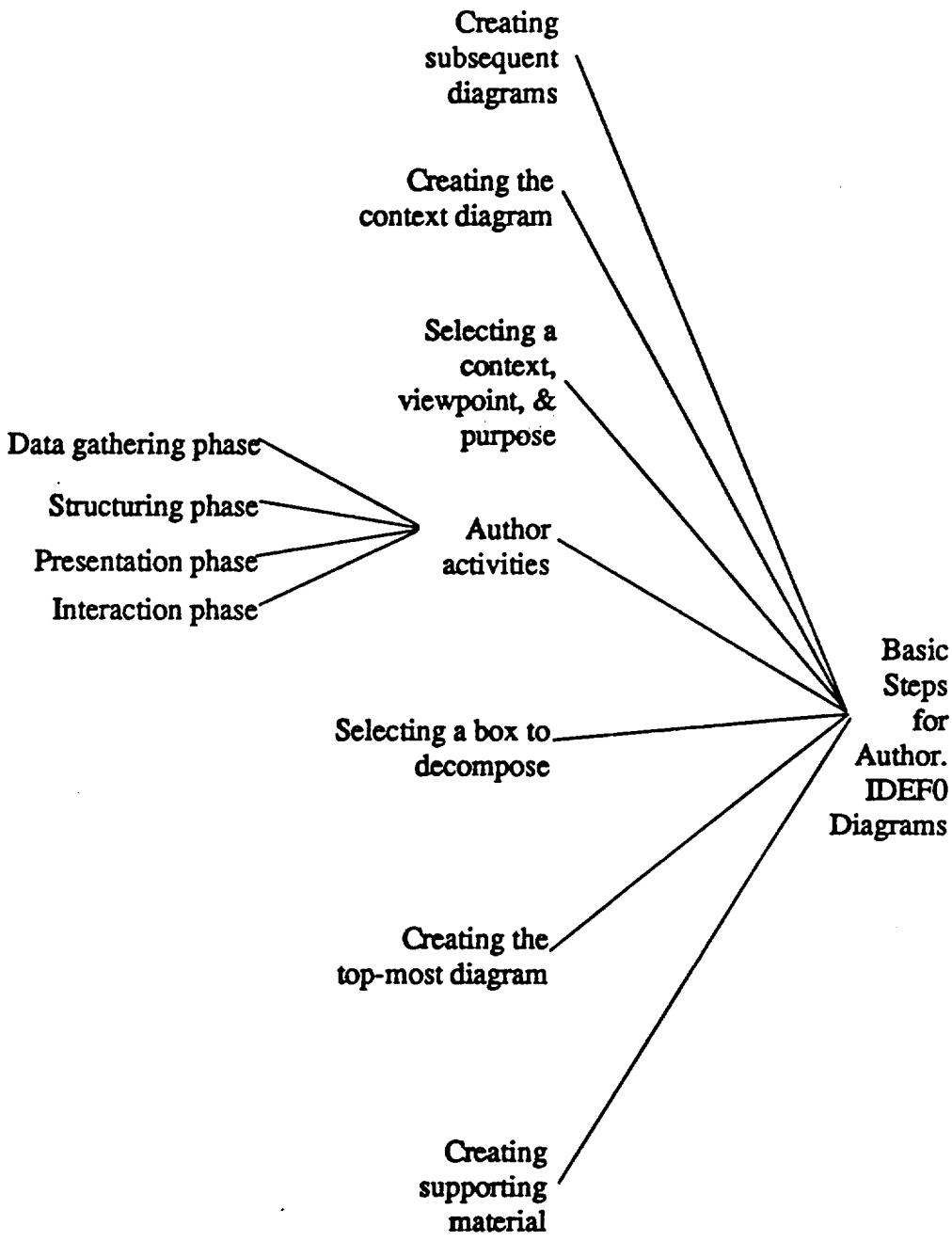


Figure B.10 IDEF0 CDG Basic Authoring Steps 801

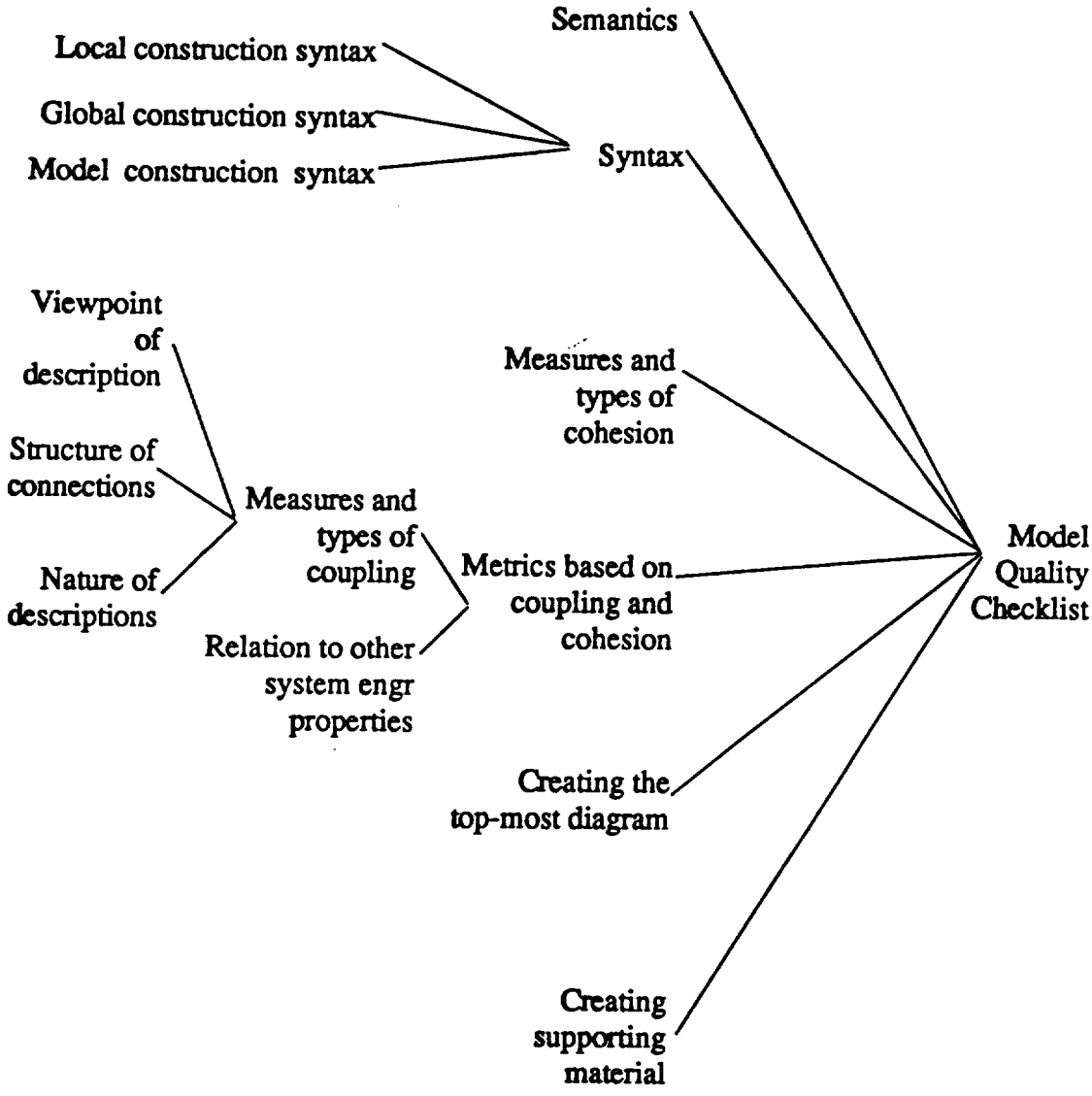


Figure B.11 IDEF0 CDG Model Quality Checklist 807