N91-10615

A Practical Approach to ObJect Based Requirements Analysis

Daniel W. Drew and Michael Bishop

Unisys, Houston Operations Division
600 Gemini Mail Code UO4C
Houston, Tx. 77058-2775
(713)-282-3664

Introduction

In the teaching of mathematics, problem statements are often used to provide exercises which require the students to apply the knowledge learned. The student must read a paragraph and determine first what the problem is, then apply the appropriate equation to find the answer. System development is analogous to solving math problem statements. There is the problem statement (requirements) which must be understood so that the right equation (design) can be applied for the solution.

If the study of mathematics emphasizes only the study of equations and how they are derived, the student will be ill-equipped to use that knowledge in practical applications. Similarly, design methods which do not have supporting methods for understanding requirements will prove difficult to use in practical system development.

The use of objects in design methodologies has provided a mechanism whereby software engineers can take fuller advantage of software engineering principles. However, these concept are just beginning to reach their full potential as we move them earlier into the lifecycle.

This paper presents an approach, developed at the Unisys Houston Operation Division, which supports the early identification of objects. This "domain oriented" analysis and development concept is based on entity relationship modeling and object data flow diagrams. These modeling techniques, based on the GOOD methodology developed at the Goddard Space Flight Center [4], support the translation of requirements into objects which represent the real-world problem domain. The goal is to establish a solid foundation of understanding before design begins, thereby, giving greater assurance that the system will do what is desired by the customer. The transition from requirements to object oriented design is also promoted by having requirements described in terms of objects.

Presented is a five step process by which objects are identified from the requirements to create a problem definition model. This process involves establishing a base line requirements list from which an object data flow diagram can be created. Entity-relationship modeling is used to facilitate the identification of objects from the requirements.

The paper concludes with an example of how semantic modeling may be used to improve the entity-relationship model and a brief discussion on how this approach might be used in a large scale development effort.

A Practical Approach to Object Based Requirements Analysis

1.0 Approach Overview

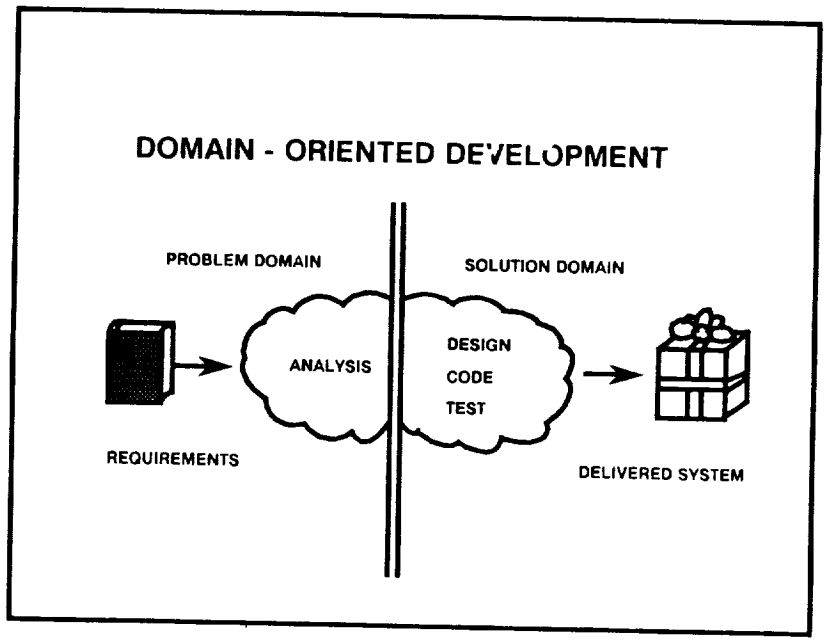
Following the principles of software engineering promotes a more pragmatic approach for system development. It requires a change in the overall concepts of how systems are created as well as new analysis and design methodologies.

1.1 Domain Oriented Development

For a design to be successful, there must be an understanding of the problem it is intended to solve. All too often problem definition is established in just enough detail to begin design and evolves as the design evolves. This can lead to unstructured systems which are hard to implement and expensive to maintain. To eliminate this problem software development can be divided into the problem and solution domains. The problem domain provides the foundation for all solution domain activities. A greater discipline is introduced into development giving greater assurance that the requirements (problem) are understood before a design (solution) is attempted.

Activities included in the problem domain are requirements generation and requirements analysis. The end product of requirements analysis is a problem definition model. This model becomes the foundation for all solution domain activities.

Activities included in the solution domain are preliminary design, detail design, code, and test. The end product is a delivered system which conforms to requirements.



1.2 The Mechanics of Requirements Analysis

Requirement analysis is concerned with establishing what a system is to do. This information must be documented in a form easily understood by all parties involved in development. The process for understanding a set of requirements requires an ordered set of steps which clarify original requirement statements and allow key information to be identified.

D. Drew
Unisys

The remainder of this paper will show in detail an approach which is made up of the following steps:

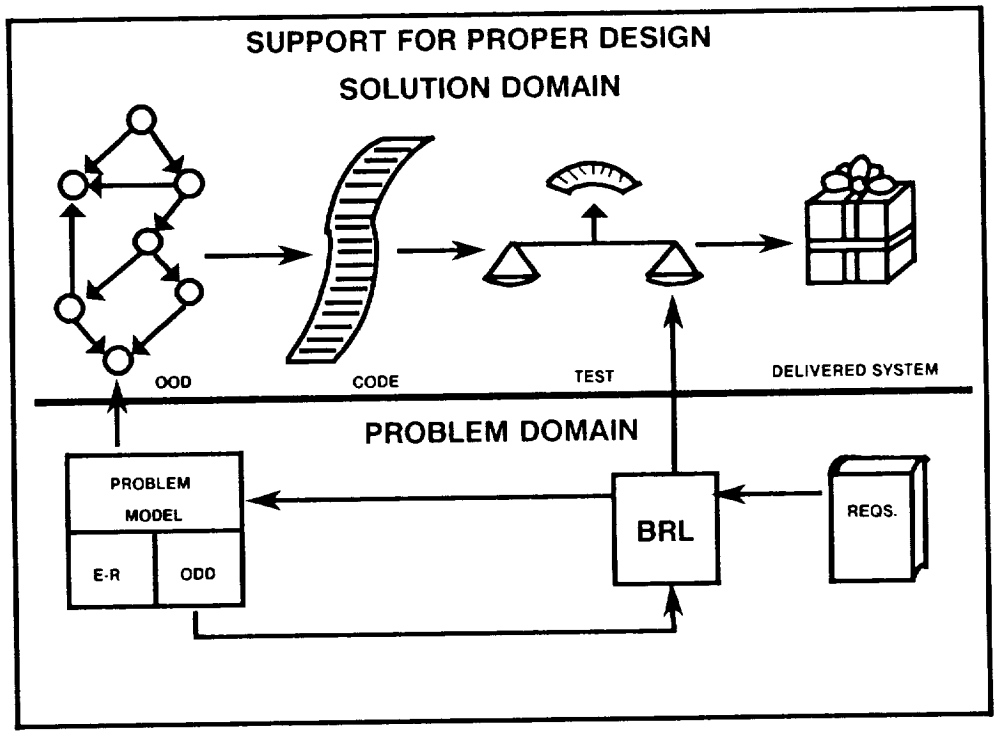
Step 1: Compile a notebook containing all requirement statements and information from other sources which might be pertinent to the problem.

Step 2: Rewrite the information from the notebook into concisely stated sentences. This establishes a baseline requirements list (BRL).

Step 3: Develop a static model of the problem from the BRL using entity relationship modeling. This model will facilitate the identification of objects.

Step 4: Identify the objects and develop a dynamic model of the problem from the entity relationship model using an object data flow diagram (ODD).

Step 5: Reorganize the BRL so that the statements are grouped by object.



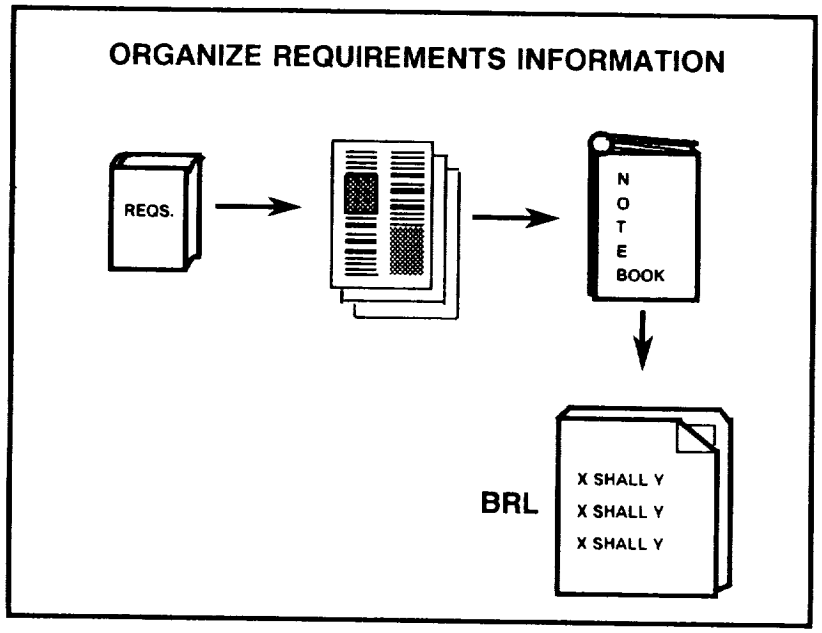
2.0 Step 1: Compiling an Information Notebook

Complete requirements information is essential in order to create the system the customer really wants. The purpose of step one is to gather all information which might have any possible bearing on what the system is to do. The actual process for this step will vary as to the sources of information available. For entirely new development projects, this is the initial step of requirements generation. Information must be compiled from many different sources. For enhancements to an existing system, this step is the identification and collection of requirements pertinent to the enhancements. Information sources would be the existing requirements document, design specifications, and interviews of current users and maintenance personnel. The end result is to have all the information available for design gathered in a single reference.

This approach was developed for a project which had an existing requirements document. The document was old and the system had undergone several major revisions. The notebook contained pages from the requirements document, information from a system closely related to the one being redesigned, and notes given by experts in the application. The end result was a collection of all available requirements information which then served as a single reference for analysis and design.

3.0 Step 2: Establishing a Baseline Requirements List

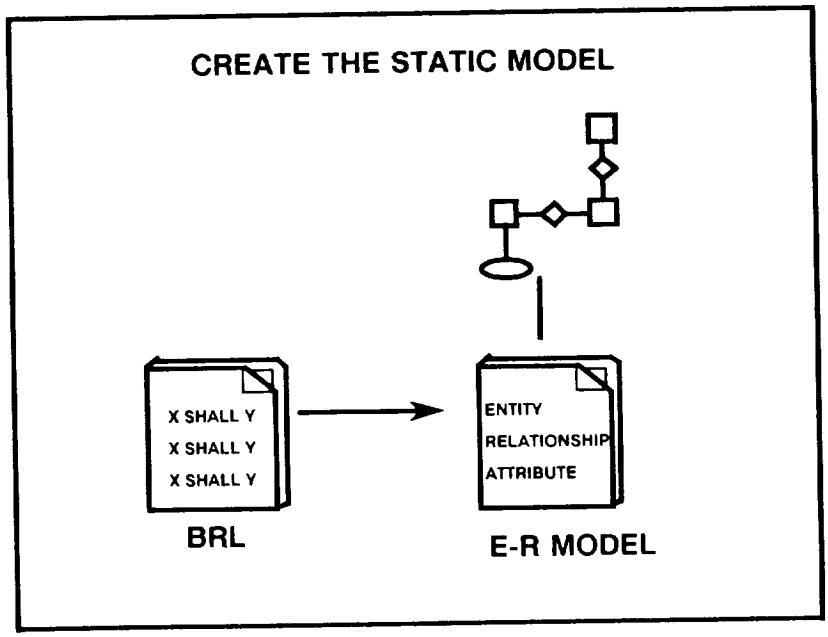
The resultant notebook contains all the information needed to create a model of what the system is to do. However, there is no meaningful structure. It is very difficult to determine: if the information is complete, if there is information that is not needed, or how the pieces of information relate to each other. A good organization of requirements is necessary in order to facilitate the extraction of entities, relationships, and attributes from the requirements and to develop the dynamic problem domain definition. The Baseline Requirements List (BRL) provides this needed structure. Each statement in the notebook is rewritten in a traditional "X shall Y" format where "X" is a noun or noun phrase and "Y" is some action the noun will perform. Rewriting in this form will force a greater understanding of each requirement piece. Ambiguous statements and statements which have no impact on what the system is to do can be easily recognized. Having all requirements stated as cause and effect also provide a solid platform for system testing.



4.0 Step 3: Developing a Static Problem Definition Model

A static model of the problem is the first component of the problem definition model. Its purpose is to give structure to the requirements information that will facilitate the identification of the dynamic properties of the system. A static model represents all the possible entities, with their attributes and relationships, described by the BRL. The development of a static model based on the requirements is an information representation problem. Therefore, it is reasonable to borrow modeling techniques from the DBMS world. Entity relationship modeling has been recommended by Mike Stark and Ed Seidewitz of the Goddard Space Flight Center [4] and Dr. Charles McKay of the University of Houston at Clear Lake [2] as an appropriate tool for the structuring of requirements information.

Issues of completeness in requirements can be addressed with this model. Incomplete requirements appear as dangling entities which have no relationships or as relationships without clearly defined entities. An entity without relationships may also indicate a requirement statement which does not belong to the problem. This type of inconsistency is identified and resolved in an iterative process of reviewing the requirement statements which make up the part of the entity-relationship model in question until all unusual model structures are resolved.



4.1 Entity-relationship Modeling

The approach promoted by this paper for entity-relationship modeling consists of the entity-relationship model creation phase, the entity dictionary, which provides entity definitions which will be used throughout the software lifecycle, and entity-relationship diagrams, which can be used to graphically depict portions of the entity dictionary. Object data-flow diagrams, which depict the dynamic problem definition are generated from the entity-relationship model and will be addressed in section 5.0. The remainder of this section presents in detail how an entity-relationship model is developed from the BRL.

A common example, a subset of a student registration system, will be presented with most of the topics in this section and in section 5 in order to help in understanding the concepts. The example will have the following requirements:

1. The system shall provide the capability to enter and maintain information regarding students.

2. The system shall provide the capability to enter and maintain information regarding the courses in which students are enrolled.

3. Student information shall include the student's name, age, major and social security number.

4. Course information shall include the course's name, department, room number, meeting time and days, name of the professor teaching the course, a list of students enrolled in the course, the number of students currently enrolled in the course and the maximum number of students allowed in the course.

5. A course shall be closed when the number of currently enrolled students reaches the maximum number of students allowed in the course. Otherwise, the course shall be considered open.

6. Students shall be allowed to enroll in an open course.

7. Students shall not be allowed to enroll in a closed course.

8. The system shall accept registration requests containing the name of a student and the name of the course in which he/she wishes to enroll.

9. Registration requests shall be processed in order to determine whether or not a student may enroll in the requested course.

4.2 Entity-Relationship Model Creation

The entity-relationship model creation phase consists of extracting entities, attributes and relationships from the requirements. During this phase, the requirements are assumed to be in the form of the BRL discussed in section 3.0.

4.2.1 Entity Extraction

Entities will appear as nouns in the requirement statements. Different types of noun phrases reveal different types of entities [3]. Common nouns, such as "terminal", "student" or "message", name a class of entities. Mass nouns and units of measure, such as "water", "matter" or "fuel", name a quality, activity, quantity or substance of the same. Proper nouns and nouns of direct reference, such as "my terminal", "George" or "syntax error advisory message", name specific instances of an entity class.

The requirements will not necessarily name all of the entities in the problem domain. Related entities may have to be found by looking through documentation, talking to people who have some expertise in the area, etc. For example, suppose that the problem domain consists of a bucket containing different types of fruit. The requirements may state that the job is to remove the apples and oranges from the bucket and place them in different piles. The entities in this problem domain, as shown by the requirements, are the apples, oranges and the bucket. However, there are other kinds of fruit that have to be considered when removing the apples and oranges (i.e. they must be discarded). Those other fruits are part of the problem domain and therefore are entities in the problem domain.

There is another case in which entities are not explicitly named in the requirements. Suppose that the requirements in the apples and oranges problem also state that someone is to be notified when a spoiled apple is found in the bucket. This new requirement introduces two new entities, a spoiled apple and a notification that a spoiled apple has been found. There is a gap in the problem domain model between the spoiled apple and the notification of the spoiled apple. This gap is filled by an entity that represents the event that is characterized by finding the spoiled apple. The event entity is related to the notification entity in that someone is to be notified in the event that a spoiled apple is found.

Entities are either internal or external. Internal entities have an existence only within the scope of the problem domain. External entities have an existence outside the scope of the problem domain. The concept of internal and external entities is easier to consider if the problem domain is thought of as a "black box." Internal entities cannot be seen outside of the box but external entities can be seen entering or leaving the box.

In the student registration example, the requirements yield the following entities:

From requirement 1: Student

From requirement 2: Course, Student

From requirement 3: Student

From requirement 4: Department, Professor, Course_Roster, Course

From requirement 5: Course, Closed_Course, Open_Course, Student

From requirement 6: Student, Open_Course

From requirement 7: Student, Closed_Course

From requirement 8: Registration_Request

From requirement 9: Course, Registration_Request, Student

The Course_Roster in requirement 4 is the list of students enrolled in a course.

4.2.2 Attribute Extraction

Attributes usually appear in the requirements as information concerning entities. The following attributes are named in the requirements:

Student:	Student_Name, Age, Major, SS_Number
Course:	Course_Name, Current_Size, Max_Size, Time, Days,Room_Number,Professor_Name, Department_Name
Professor:	Professor_Name
Department:	Department_Name
Registration_Request:	Student_Name, Course_Name

4.2.3 Relationship Extraction

Relationships appear in the requirements as associations between pairs of entities, entities and attributes or relationships and attributes. The student registration requirements show the following relationships:

Requirement 2:	Is_Enrolled_In (1:m) between Student and Course.

Requirement 4: Includes (1:1)/Is_A_Part_Of (1:1)
between Course and Course_Roster;
Includes (1:m)/Is_A_Part_Of (m:1)
between Department and Course, Professor;
Is_A_List_Of (1:m)/Is_A_Member_Of (m:1)
between Course_Roster and Student;
Teaches (1:1)/Is_Taught_By (1:1)
between Professor and Course.

Requirement 5: Is_A_Type_Of (1:1)
between Closed_Course and Course,
between Open_Course and Course;
Is_An_Instance_Of (1:1)
between Course and Open_Course or Closed_Course.

Requirement 6: May_Enroll_In (1:m)
between Student and Open_Course.

Requirement 7: May_Not_Enroll_In (1:m)
between Student and Closed_Course.

Requirement 9: References (1:1) / Is_Referenced_By (1:m)
between Registration_Request and Student,
between Registration_Request and Course.

A slash between two relationship names indicates a pair of symmetric, oppositely-directed relationships. In requirement 4, Course includes Course_Roster and conversely, Course_Roster is a part of Course. The mapping class of the relationship is indicated in parentheses.

4.3 Entity Dictionary

The entity dictionary provides a means of describing the entities that are part of the problem domain. A data structure that is useful for representing the entity dictionary is the frame [4], a form of knowledge representation developed by Marvin Minsky. A frame is a generalized property list containing a list of symbols with their associated property names and values [5].

The following is an example of entity entries in the student registration entity dictionary.

Closed_Course (Entity)
    Rqmt_Numbers 5, 7
    Scope External
    Is_A_Type_Of Course

Course (Entity)
    Attributes Course_Name, Department_Name, Room_Number, Time, Days,
                Professor_Name, Current_Size, Max_Size
    Rqmt_Numbers 2, 4, 5, 9
    Scope External
    Is_Taught_By Professor
    Is_A_Part_Of Department
    Is_An_Instance_Of Open_Course, Closed_Course

Is_Referenced_By Registration_Request
Includes Course_Roster

Course_Roster (Entity)
Rqmt_Numbers 4
Scope Internal
Is_A_Part_Of Course
Is_A_List_Of Student

Department (Entity)
Attributes Department_Name
Rqmt_Numbers 4
Scope External
Includes Course, Professor

Open_Course (Entity)
Rqmt_Numbers 5, 6
Scope External
Is_A_Type_Of Course

Professor (Entity)
Attributes Professor_Name
Rqmt_Numbers 4
Scope External
Teaches Course
Is_A_Part_Of Department

Registration_Request (Entity)
Attributes Student_Name, Course_Name
Rqmt_Numbers 8, 9
Scope Internal
References Student, Course;

Student (Entity)
Attributes Student_Name, Age, Major, SS_Number
Rqmt_Numbers 1, 2, 3, 5, 6, 7, 9
Scope External
Is_Enrolled_In Course
Is_A_Member_Of Course_Roster
May_Enroll_In Open_Course
May_Not_Enroll_In Closed_Course
Is_Referenced_By Registration_Request

The entity dictionary can be extended to include attributes. The following is an example of some of the attribute entries in the student registration entity dictionary.

Course_Name (Attribute)
Is_An_Attribute_Of Course
Rqmt_Numbers 4, 8
Domain String

Days (Attribute)
Is_An_Attribute_Of Course
Rqmt_Numbers 4

Domain Character
Values M, T, W, R, F, MWF, TR, MW

Student_Age (Attribute)
Is_An_Attribute_Of Student
Rqmt_Numbers 3
Domain Integer
Range 16..100

Time (Attribute)
Is_An_Attribute_Of Course
Rqmt_Numbers 4
Domain Character
Length 5
Range 08:00..19:00

4.4 Entity-Relationship Diagrams

Entity-relationship diagrams are used to graphically depict a part of the problem domain. Attempts were made to split the problem domain into parts by using a levelling technique in which the upper levels in the problem domain consist of "aggregate entities" with the actual problem domain entities at the lower levels. Unfortunately, there was not much progress in this endeavor and therefore a single-level description of the problem domain was created. Since a diagram showing the entire problem domain would be cumbersome, it is better to use the entity dictionary as the problem domain definition with entity-relationship diagrams being generated to map parts needing greater clarification. [4].

In the entity-relationship diagram, entities are represented by rectangles and relationships by diamond-shaped boxes [1]. Attributes are listed next to the rectangle representing the entity. The arrows indicate the direction of relationships. A double-h aded arrow indicates the 1:m, m:1 or m:n mapping class.

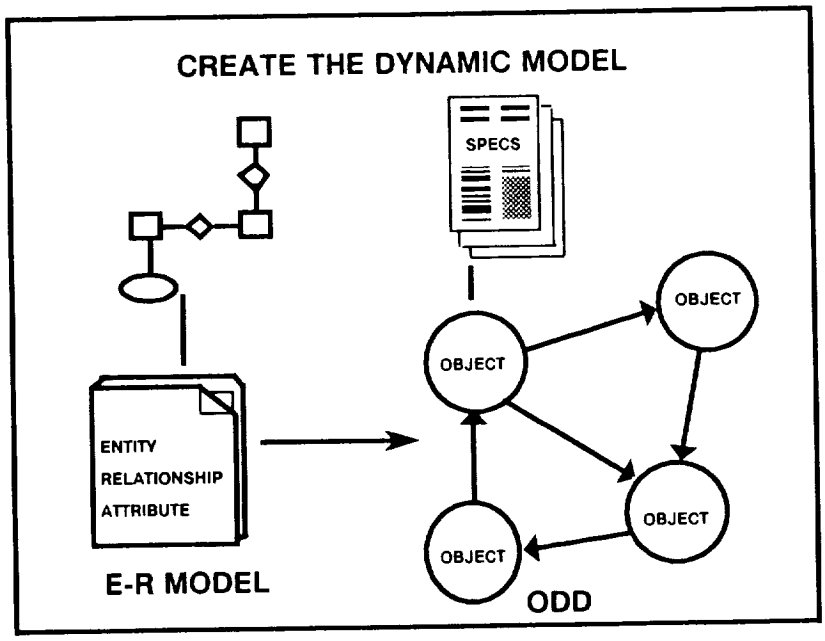
Entity-relationship diagrams can be generated in order to graphically map the problem domain onto one or more requirements or to show the problem domain from the perspective of a particular entity. In the latter application, it is useful to state the "order" of the diagram. A first-order entity-relationship diagram shows the central entity (the entity from whose perspective the problem domain is being viewed) and its relationships to surrounding entities. A second-order diagram shows the central entity, its relationships to surrounding entities and the relationships of each of the surrounding entities to its surrounding entities.

Sample entity-relationship diagrams for the student registration system are shown in appendix A.

5.0 Step 4: Developing a Dynamic Problem Definition Model

The second and concluding component of the problem definition model is the dynamic model of the problem. It is through this model that data flow and control, as described by the requirements, is represented. An object data flow diagram (ODD) is used to model the dynamic properties of the problem [4]. An ODD is very similar to a data flow diagram from Yourdon structured analysis techniques. The chief difference lies in what the bubbles represent. For an ODD the bubbles are objects. Since data is encapsulated in objects, there will not be any data stores.

The remainder of this section will present in detail how an ODD is derived from an entity-relationship model.



5.1 Identifying Problem Domain Objects

An object is a unique instance of an abstract data type which is a set of data and operations associated with that data. In order to identify the objects in the problem domain, first find all of the major abstract data types apparent in the problem domain and use an object to manage each one. The abstract data types are represented by entities that do not have an Is_A_Type_Of, Is_An_Instance_Of, Is_A_List_Of or Is_A_Set_Of relationship to another entity. These entities are at the highest level of abstraction for entities of a particular type. In the student registration problem, those entities are Course, Department, Professor, Registration_Request and Student. Each one of these entity classes will have an object to manage it. These candidate objects are Course_Folder, Department_Folder, Professor_Folder, Registration and Student_Folder.

The next step is to find all entity classes associated through the Is_A_Type_Of, Is_A_List_Of or Is_A_Set_Of relationship with the entity classes found in the first step. In the example, Open_Course and Closed_Course are associated with Course through the Is_A_Type_Of relationship and Course_Roster is associated with Student via the Is_A_List_Of relationship. The following objects and associated entities can be identified thus far:

Course_Folder:	Course, Open_Course, Closed_Course
Department_Folder:	Department
Professor_Folder:	Professor
Registration:	Registration_Request
Student_Folder:	Student, Course_Roster

The entity classes listed for each object represent the abstract data types provided by that object. At this point, it is possible to determine the set of requirements satisfied by each object. This is done by consulting the entity dictionary and finding the requirement statement numbers for each of the entities associated with each object. Applying this process to the five objects in the student registration system shows the requirements satisfied by each object:

Course_Folder:	Requirements 2, 4, 5, 6, 7 and 9
Department_Folder:	Requirement 4
Professor_Folder:	Requirement 4
Registration:	Requirements 8 and 9
Student_Folder:	Requirements 1, 2, 3, 4, 5, 6, 7 and 9

The requirement sets for each object are not disjoint. The shared requirements (numbers 2, 4, 5, 6, 7 and 9 in our example) describe the relationships between entities of different types. These relationships, in turn, describe the interfaces between different objects.

In order to complete the definition of the problem domain objects, find all relationships between entities of different types and add each member of the corresponding entity pairs to the appropriate object. For example, because of the relationship "Student Is_Enrolled_In Course", there is an interaction between the Student_Folder and Course_Folder objects. To show this interaction, add the entity Student to the Course_Folder object and Course to Student_Folder. One exception to this procedure occurs when one member of the entity pair is a generalized entity class. An example of this is the relationship "Student May_Enroll_In Open_Course". Since "Open_Course Is_A_Type_Of Course", add Course to Student_Folder instead of Open_Course. This procedure results in the following set of objects and associated entities:

Course_Folder:	Course, Open_Course, Closed_Course, Course_Roster, Student, Department, Professor, Registration_Request
Department_Folder:	Department, Course, Professor
Professor_Folder:	Professor, Department, Course
Registration:	Registration_Request, Student, CourseStudent, Course_Roster, Course, Registration_Request

Having identified the problem domain objects and their associated entities, an object data-flow diagram can be generated.

5.2 Generating Object Data-Flow Diagrams

Generating an object data-flow diagram based on a set of problem domain objects is simply a matter of finding entities common to pairs of objects. For example, the entities that Course_Folder and Student_Folder have in common are Student, Course_Roster and Course. Those common entities represent interfaces between Course_Folder and Student_Folder. On the object data-flow diagram, the interfaces are represented by drawing a line between the two rectangles representing the objects and labeling the line with the names of the common entities. The object data-flow diagram representing the objects from section 5.1 is in Appendix B.

The problem domain objects identified can be formally documented (in terms of the entities used and produced) by adding them to the entity dictionary:

Course_Folder (Object)
Rqmt_Numbers 2, 4, 5, 6, 7, 9
Uses Department, Course_Roster, Student, Registration_Request, Professor
Produces Course

Department_Folder (Object)
Rqmt_Numbers 4
Uses Professor, Course
Produces Department

Professor_Folder (Object)
Rqmt_Numbers 4
Uses Department, Course
Produces Professor

Registration (Object)
Rqmt_Numbers 8, 9
Uses Student, Course
Produces Registration_Request

Student_Folder (Object)
Rqmt_Numbers 1, 2, 3, 4, 5, 6, 7, 9
Uses Course, Registration_Request
Produces Student, Course_Roster

5.3 Object Names

The names given to objects play a key role in the development and understanding of the ODD. Naming objects is possibly the most difficult task in requirements analysis. The objects supply the framework for the representation of information and the eventual design. Therefore, their names must convey a concise meaning of the abstraction.

Object names are always nouns or noun phrases. This facilitates using the objects as a structure which can be used to explain action. It should be obvious from the name what real world object is represented. It is very difficult if not impossible to pick object names which do not bias design toward a particular direction. Therefore, this fact must be understood and preconceived notions must be addressed when a name is chosen. The name must be broad enough to encompass all the details associated with an object. Operations found within an object should not contradict the implied meaning of the object's name.

6.0 Step 5: Reorganization of the BRL

The entity-relationship model and ODD provide a complete problem definition model. Furthermore, the ODD serves as a platform to launch into an object oriented design. The last step for the problem domain segment of development is to go back and group the statements in the BRL under headings which represent the objects they support.

The objects are the main organizational structure for the system. Re-grouping the requirements will help the designers to find the additional detail needed to continue development. It will help the testers create test procedures aligned along object boundaries. It will simplify the traceability of requirements to design for the designer, tester, and maintainer. In short, having the requirements document reflect the structure of the emerging design will provide a high level of continuity throughout the system's lifecycle.

7.0 Enhancements to Problem Definition Modeling

Requirements analysis is a specific application of an information representation problem. As current modeling techniques evolve, it is reasonable to expect improvements in the approach taken in problem definition modeling. Semantic data models are currently being introduced for use in modeling data bases. They provide a richer medium for the representation of information. This section describes how semantic modeling can be used to enhance the entity relationship model.

Semantic data models allow designers to represent the entities of interest in an application in a way that more closely resembles the view the user has of these entities [6]. Semantic data models provide abstraction constructs that can be used to capture some of the meaning of the user application.

The semantic entity-relationship model introduced in this section features the abstraction constructs provided by the semantic and hyper-semantic [6] data models and allows the analyst to further define the problem by stating the meaning of relationships between entities in the problem domain.

7.1 Modeling Primitives

Modeling primitives are atomic relationships whose meanings cannot be defined as a composition of other meanings. They form the basis on which other relationships can be defined. Modeling primitives can be grouped into relationship classes which correspond to the abstraction constructs of the hyper-semantic data model. The modeling constructs of the hyper-semantic data model and their associated relationship names include [6]

Generalization: Similar entities are abstracted into a higher level entity-class. Relationship: Is_A_Type_Of.

Classification: Specific instances are considered as a higher level entity-class. Relationship: Is_An_Instance_Of.

Aggregation: An entity is related to the components that make it up. Relationship: Includes / Is_A_Part_Of.

Set Membership: Several entities are considered as a higher level set entity-class.Relationships: Is_A_Set_Of / Is_A_Member_Of.

List Membership: Several entities are considered as a higher level list entity-class. Relationships: Is_A_List_Of / Is_A_Member_Of.

Constraint: A restriction is placed on some aspect of an entity or relationship. Relationship: Is_A_Constraint_On.

Heuristic: An information derivation mechanism is attached. Relationship: Is_A_Heuristic_On.

Synchronous Temporal: Specific entities are related by synchronous characteristics and considered as a higher-level entity-class. Relationships: Is_A_Predecessor_Of / Is_A_Successor_Of.

Asynchronous Temporal: Specific entities are related by asynchronous characteristics and considered as a higher-level entity-class. Relationships: Initiates / Is_Initiated_By.

Equivalence: Specific instances of an entity-class are asserted to be equivalent. Relationships: Is_Equivalent_To.

The slash within the relationship names indicates two oppositely-directed relationships.

7.2 Semantic Relationship Definition

The semantic entity-relationship model provides a construct that allows the analyst to define the meaning of a relationship. This construct can be used to define a relationship in terms of other relationships and modeling primitives and to define the restriction class of a relationship.

A relationship between entity classes A and B is restricted if instances of type A may only be related to certain instances of type B based on a condition. The relationship is existence restricted if instances of type A may only to be related to those instances of type B for which they depend on their existence [7].

In order to walk through a short example of a relationship definition, consider the Is_Enrolled_In relationship between Student and Course. The objective is to state what is meant by the phrase, "a student is enrolled in a course." The course roster may be used in order to determine if a particular student is enrolled in a particular course. Remember from section 4.2 that a course roster is a list of students enrolled in a course. Therefore, a student is enrolled in a course if the student is on the course roster. The relationship is written in the following form using the semantic relationship definition construct:

```
entity class Course, Student, Course_Roster;

relationship Is_Enrolled_In (entity_instance, entity_instance);

Student Is_Enrolled_In Course if
        CR Is_An_Instance_Of Course_Roster and
        Course Includes CR and
        Student Is_A_Member_Of CR;
```

The relationship statement declares Is_Enrolled_In as a relationship between two entity instances. Therefore, the definition of the Is_Enrolled_In relationship between Student and Course is concerned with an instance of Student and an instance of Course.

The first clause within the relationship definition, "CR Is_An_Instance_Of Course_Roster", defines an entity CR which is an instance of entity class Course_Roster. The second clause, "Course Includes CR", associates CR with the particular instance of Course with which the relationship is invoked. The third clause states that the instance of Student with which the relationship is invoked must be a member of the course roster CR in order for the Is_Enrolled_In relationship to be satisfied.

The relationship is invoked by replacing Student and Course with appropriate instances, for example "George Is_Enrolled_In Physics". In this invocation of the relationship, CR is the course roster for Physics and the relationship is satisfied if George is on that roster.

The semantic relationship definition construct can be thought of as "infix Prolog". In fact, it is rather easy to convert the above example into Prolog:

```
is_enrolled_in (Student, Course):-
        is_an_instance_of (CR, Course_Roster),
        includes (Course, CR),
        is_a_member_of (Student, CR).
```

If one could "code" the modeling primitives in Prolog and generate the appropriate Prolog declarations, it would be possible to execute a problem domain model. This may be useful in ensuring that the problem domain model is correct before going on to create objects and initiate design. This process is analogous to executing a design before implementation.

8.0 Considerations For Large Projects

This paper is based on a small project projected to be only 10,000 lines of code. An important question to ask is, "How will this approach support the development of a large system of 500,000 lines or greater?"

The basic approach is good for any size project. What complicates larger systems is the large number of requirements to be considered. It may not be practical or even possible to examine all the requirements at the same time as was done for this project.

To resolve this problem, approach the requirements as layers of abstraction. Read through the document and extract those statements which define a very high level view of the system. Apply the approach presented in this paper to produce a problem definition model for this high level abstraction. Now begin an iterative process of stripping off layers of detail for each object identified in the previous level of abstraction and create a problem definition model. Use the approach presented in this paper for each iteration.

As each layer of abstraction is added to the model, check the preceding layer to assure that the objects and interfaces already established still hold true. If there are inconsistencies, make the necessary adjustments and continue with the process.

A Practical Approach to Object Based Requirements Analysis

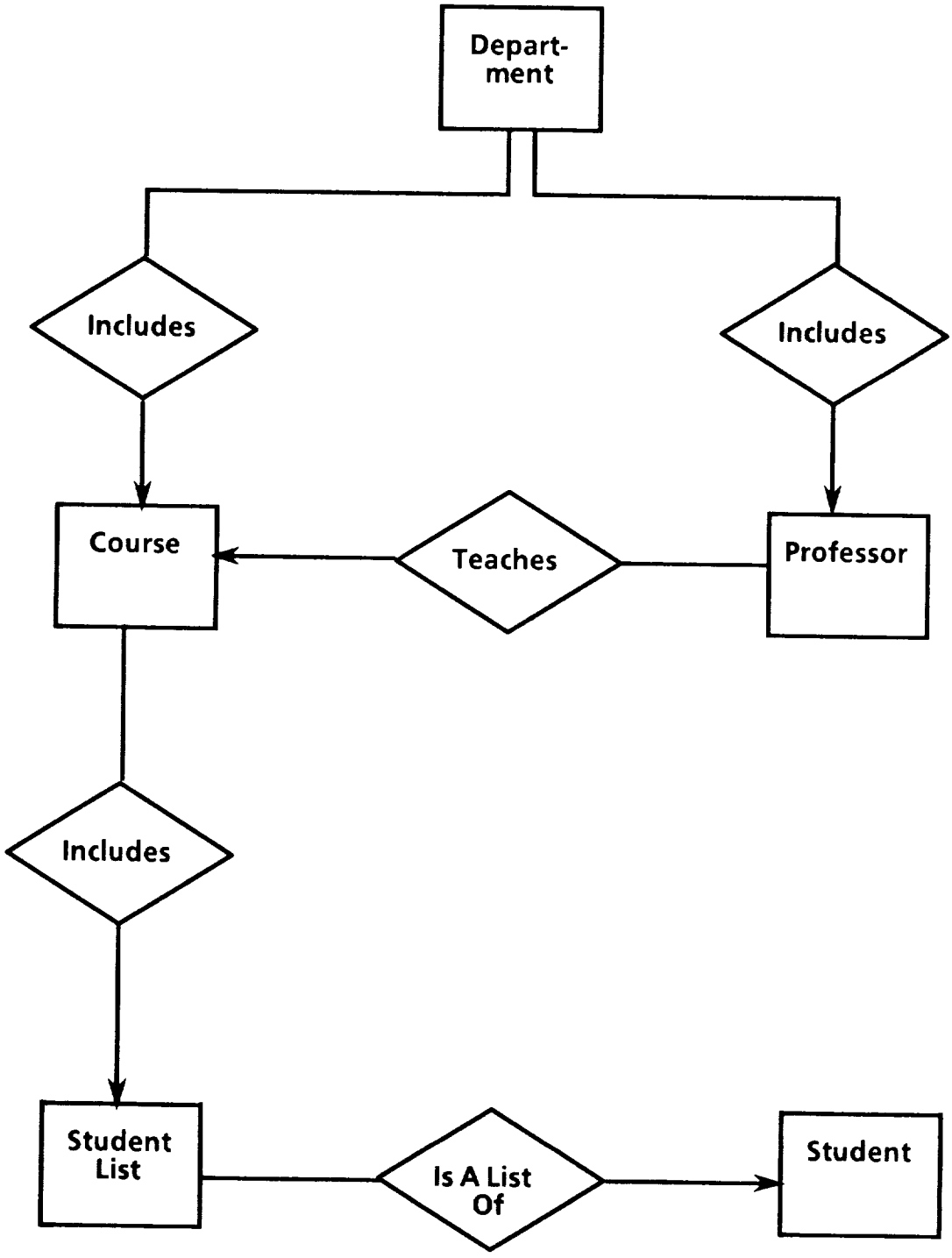
Summary

Students who spend all their time understanding math equations without applying them to problem statements will be ill-equipped to solve real world problems. System developers who possess the latest techniques in system design but have inadequate approaches to requirements analysis are destined to create wonderful designs which solve the wrong problem. The approach in this paper is a beginning to the application of modern analysis techniques rooted in the theoretical foundation of software engineering. A pragmatic approach allows for better conformance to those requirements in design. A model based on objects permits closer adherence to software engineering principles earlier in the lifecycle. It is not always easy to see objects in the requirements. Use of the entity-relationship model eases this problem by structuring the information in a form more conducive to object recognition.

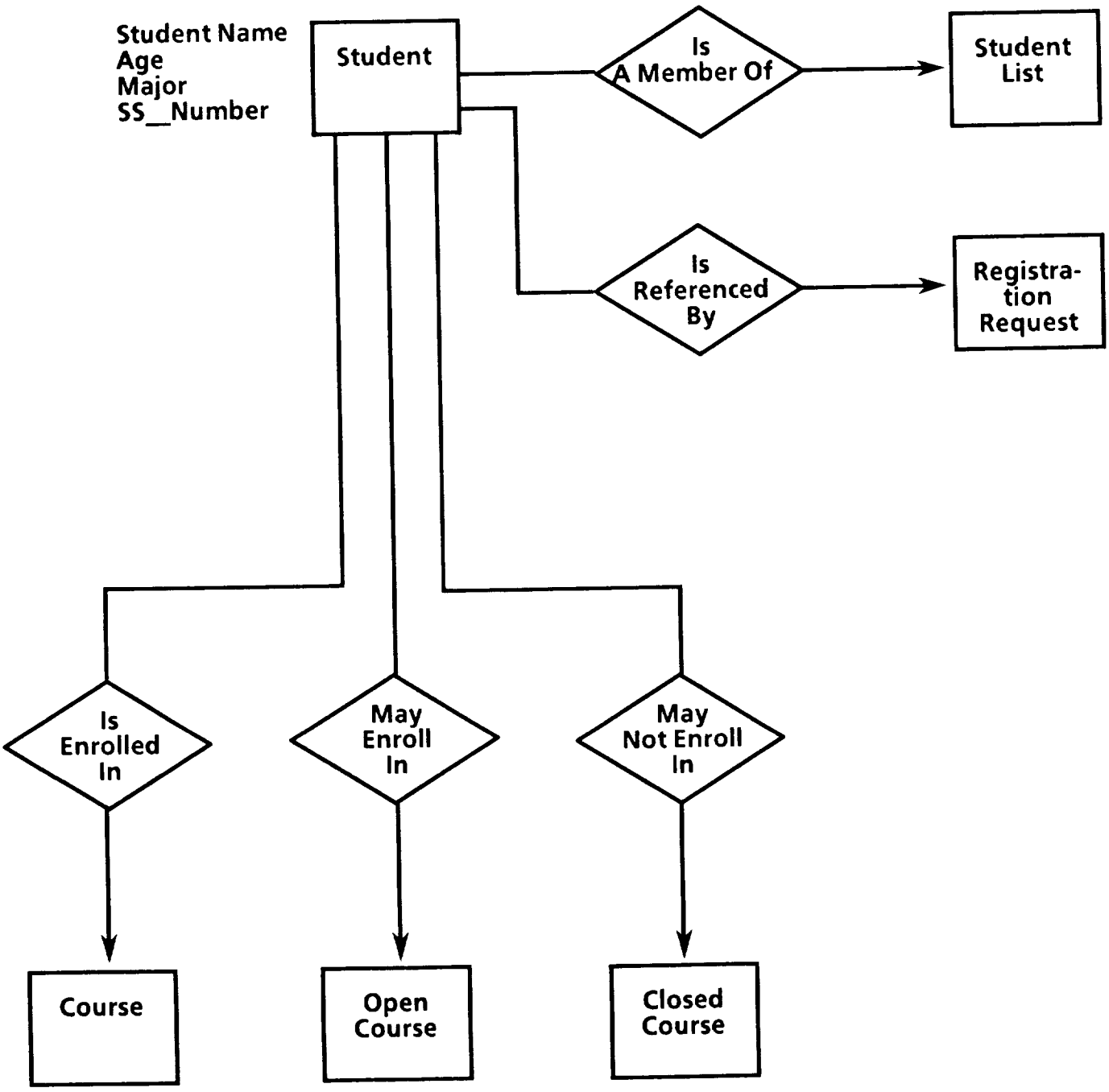
D. Drew
Unisys

Appendix A. Sample Entity__Relationship Diagrams

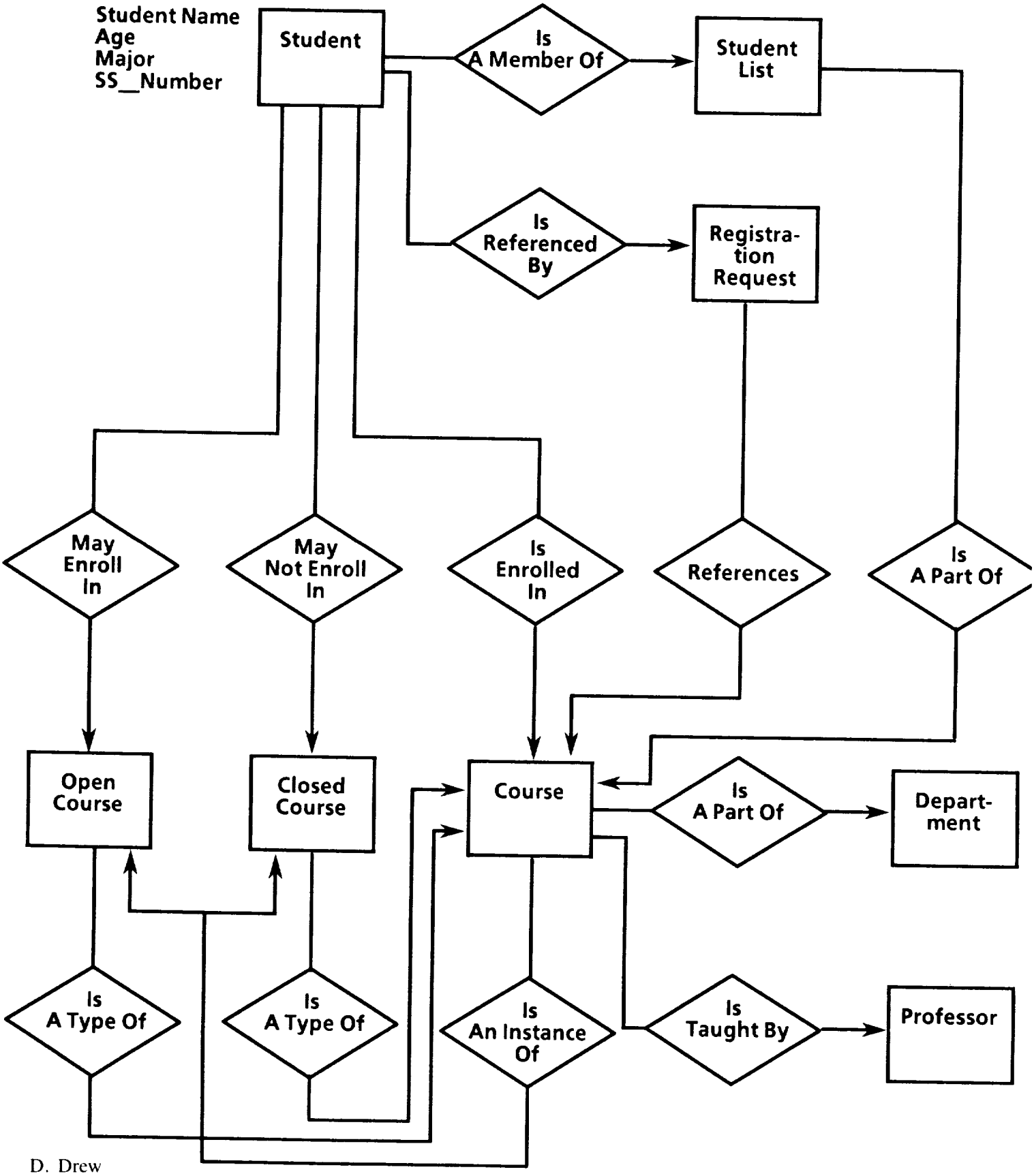
E-R DIAGRAM FOR STUDENT REGISTRATION REQUIREMENT 4



First order diagram for entity Student

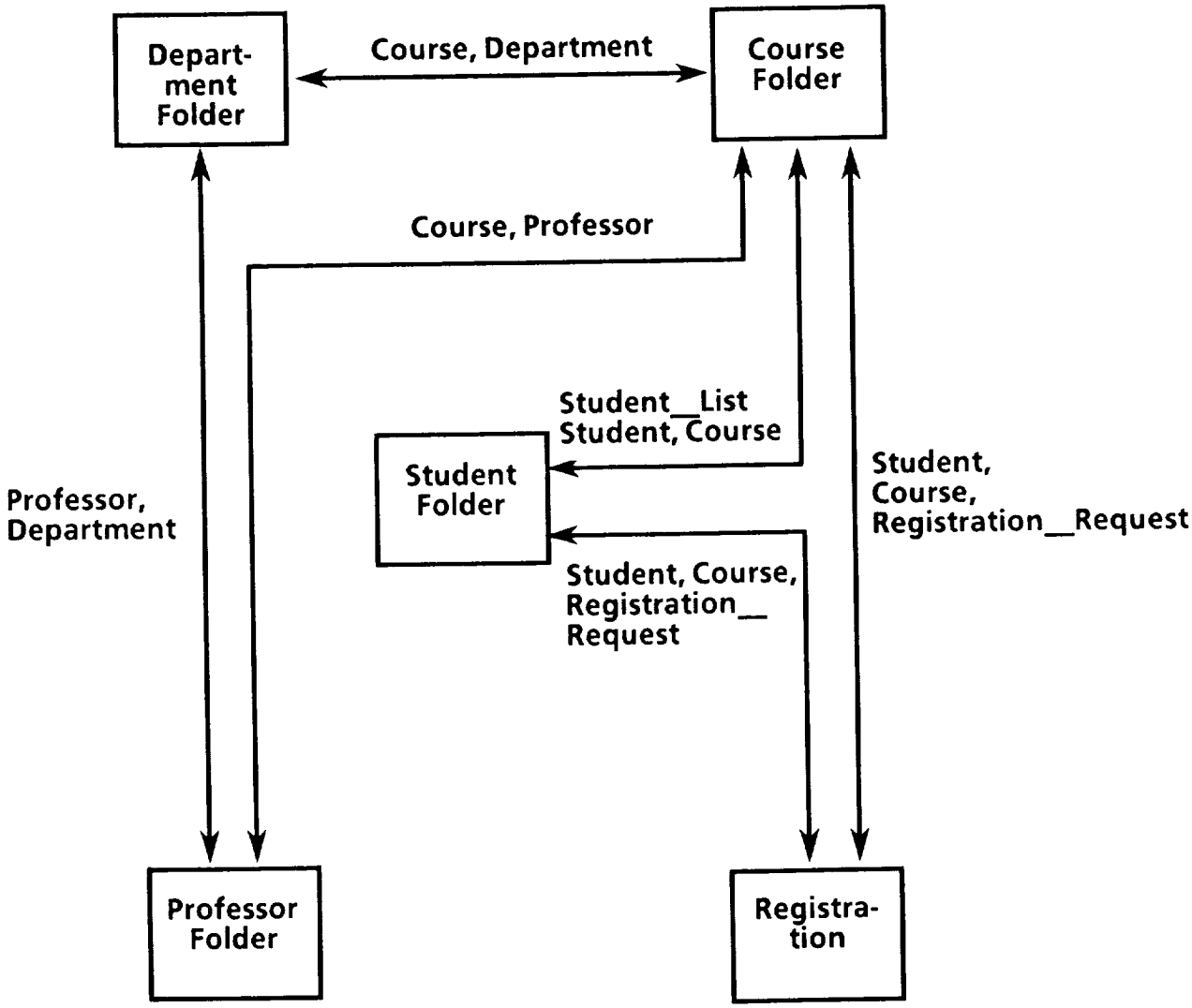


Second order diagram for entity Student



Appendix B. Sample Object Data-Flow Diagram

Object data-flow diagram for student registration system



References

[1] Chen, Peter P., "The Entity-Relationship Model - Toward a Unified View of Data", ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976, pp. 9-36.

[2] McKay, Charles W., "A Perspective and Overview of Software Engineering", a seminar sponsored by the Software Engineering Research Center at the University of Houston at Clear Lake.

[3] Booch, Grady, Software Engineering With Ada, Second Edition, The Benjamin/Cummings Publishing Company, Inc., 1987.

[4] Stark, Mike and Seidewitz, Ed, "Towards A General Object-Oriented Ada Lifecycle", Goddard Space Flight Center, Greenbelt, Md., March 1987.

[5] Winston, Patrick H. and Horn, Berthold K. P., Lisp, Addison-Wesley Publishing Company, 1981.

[6] Potter, Walter D. and Trueblood, Robert P., "Traditional, Semantic, and Hyper-Semantic Approaches to Data Modeling", Computer, June 1988, pp. 53-63.

[7] Webre, Neil W., "An Extended Entity-Relationship Model And Its Use On A Defense Project", Entity-Relationship Approach To Information Modeling And Analysis, ed. by Peter P. Chen, Elsevier Science Publishing Company, 1983.

Biographical Sketch

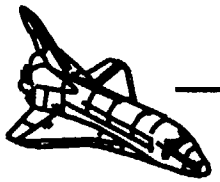
Daniel Drew has worked 13 years in the computer industry. After graduating with a B.S. degree in Computer Science from Texas A&M University, he spent 10 years developing Supervisory, Control, and Data Acquisition (SCADA) systems for oil pipeline control and automated oil field production. He has spent the last three years in Aerospace as a system designer and currently as section supervisor at the Unisys, Houston Operations Division. The section he supervises is working the first Ada pilot project attempted at the Unisys Houston site. Mr. Drew is a member of the IEEE Computer Society, Clear Lake Chapter of SigAda, and National SigAda.

Michael Bishop has worked as a programmer and systems analyst in the aerospace industry for over four years. Mr Bishop is currently employed at the Unisys Houston Operation Division where he has been developing an entity-relationship methodology suitable for a wide range of applications as part of an Ada pilot project. Previously, Mr. Bishop worked at Ford Aerospace and Unisys on the MAST project, a database application concerned with the management of Space Shuttle downlink and uplink data. Mr. Bishop received his bachelor's degree in Computer Science in 1984 from the University of Houston and is currently pursuing a master's degree at the University of Houston's Clear Lake campus.

THE VIEWGRAPH MATERIALS

FOR THE

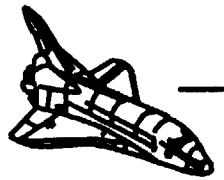
D. DREW PRESENTATION FOLLOW



UNISYS
Houston Operations

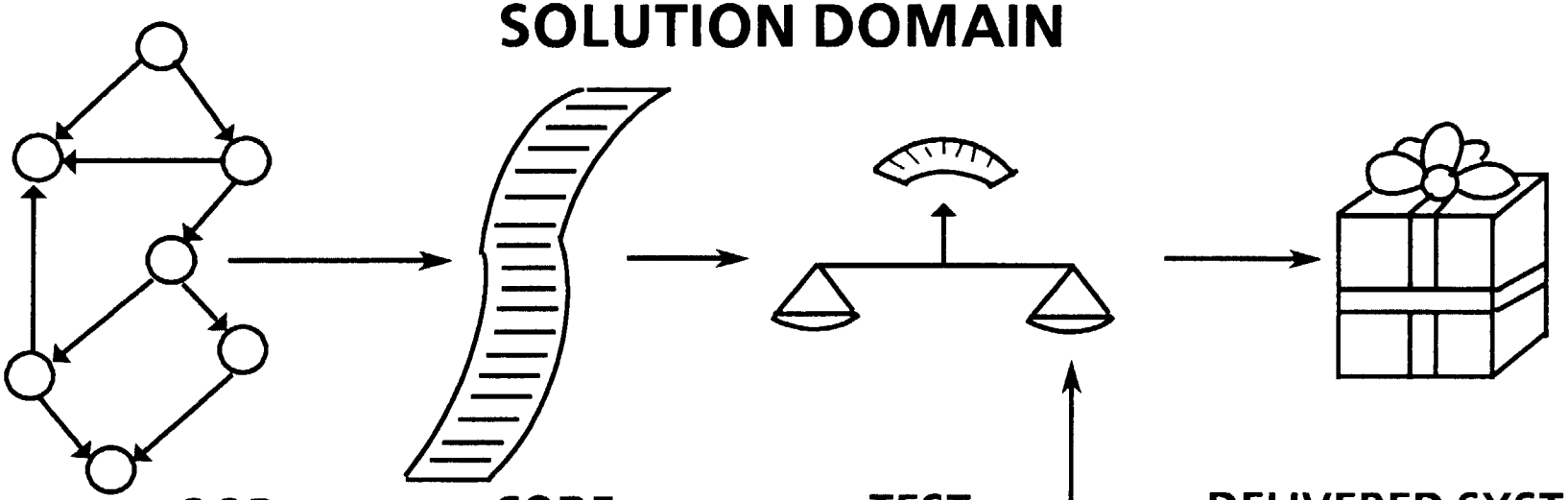
A PRACTICAL APPROACH TO OBJECT BASED REQUIREMENTS ANALYSIS

DANIEL W. DREW

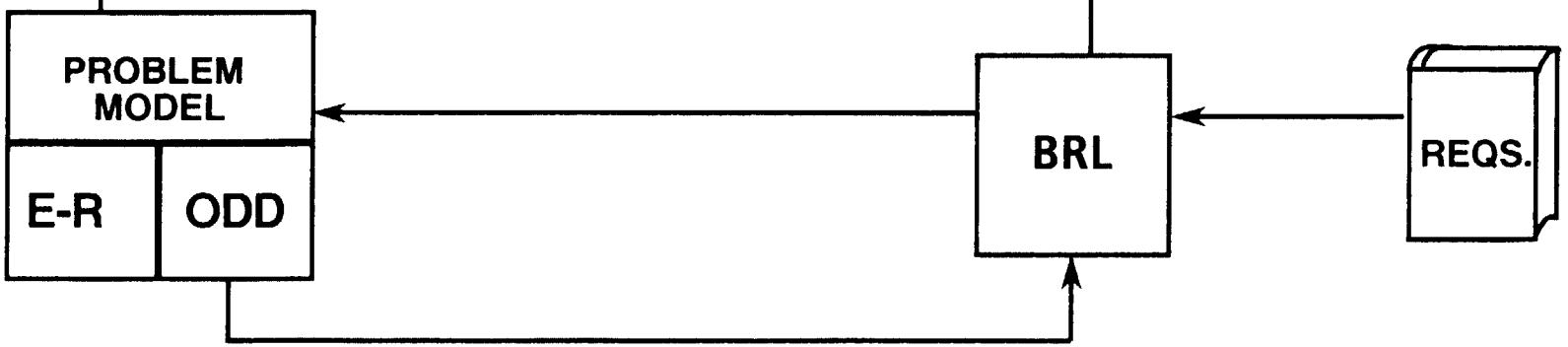


OVERVIEW OF APPROACH

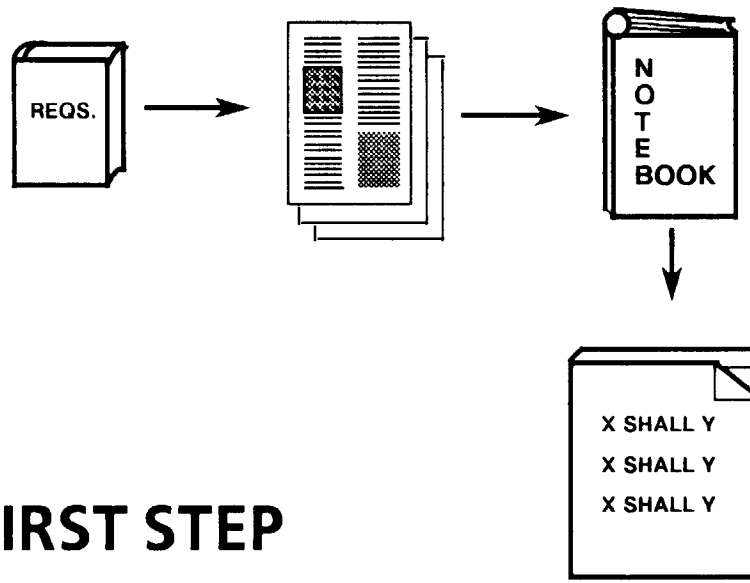
SOLUTION DOMAIN



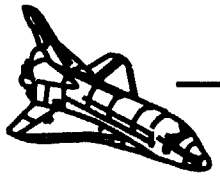
PROBLEM DOMAIN



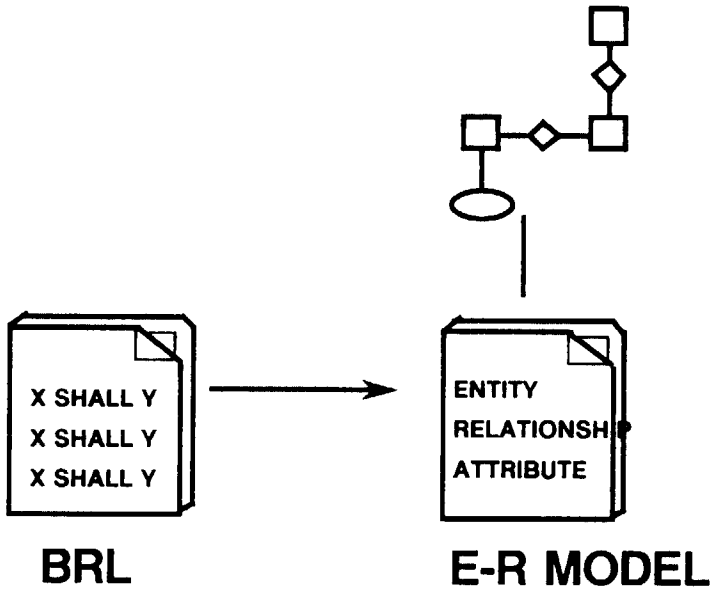
ORGANIZE REQUIREMENTS INFORMATION



- AN ESSENTIAL FIRST STEP
- FORCES EXAMINATION OF EACH REQUIREMENT STATEMENT
- AUTOMATED TRACKING SYSTEM A MUST

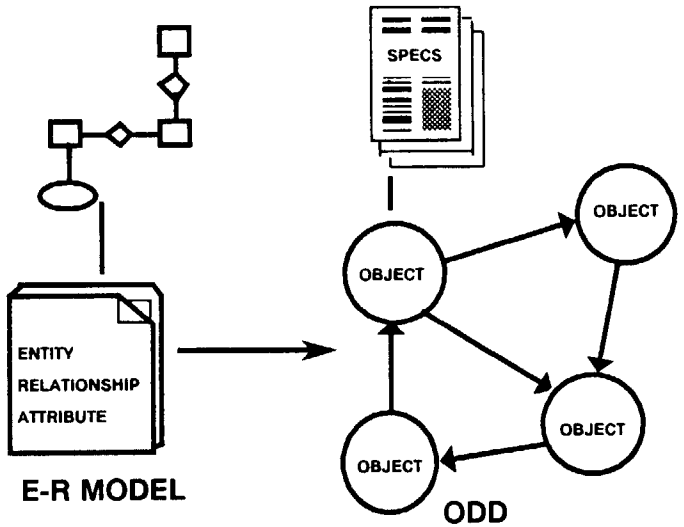


CREATE THE STATIC MODEL

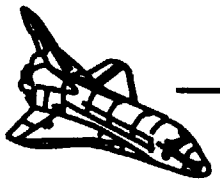


- STRAIGHTFORWARD EXTRACTION OF ENTITIES
- RELATIONSHIP/ATTRIBUTE IDENTIFICATION NOT AS CLEAR
- ENCOUNTERED DIFFICULTY IN ATTEMPT TO LEVEL E-R MODEL
- NEED FOR COMPUTER-AIDED TOOL

CREATE THE DYNAMIC MODEL



- SMOOTH TRANSITION TO OBJECTS
- NAMING OBJECTS VERY CRITICAL
- DIFFICULT TO AVOID DESIGN ISSUES
- MUST MAINTAIN DETAILED INFORMATION



SUMMARY

- PROBLEM IS TO REPRESENT INFORMATION
- USE DATABASE MODELING TECHNIQUES
- COMMUNICATION ISSUES MUST BE ADDRESSED
- CUSTOMERS EASILY UNDERSTAND OOD PRESENTATIONS