IDEF3 and IDEF4 Automation System Requirements Document and System Environment Models

Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Thomas M. Blinn of Knowledge Based Systems, Inc. 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 the 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.

The views and conclusions contained in this report are those of the author and should not be interpreted as representative of the official policies, either express or implied, of NASA or the United States Government.

IDEF3 and IDEF4 Automation System Requirements Document and System Environment Models

An Interim Technical Report

Developed By: Knowledge Based Systems, Inc.
2746 Longmire Drive
College Station, TX 77845-5424
(409) 696-7979

Principal Investigator: Thomas M. Blinn

Developed For: Artificial Intelligence Section
NASA Johnson Space Center
Houston, TX 77058

Under Subcontract to: RICIS Program
University of Houston - Clear Lake
Houston, Texas 77058-1096

Subcontract Number 055:
Cooperative Agreement Number: NCC 9-16

August 1, 1989 - November 20, 1989

Table of Contents

List of Figures

1.0 Introduction

This document provides the requirements specification for the IDEF3 and IDEF4 tools that provide automated support for IDEF3 and IDEF4 modeling. The IDEF3 method is a scenario-driven process flow description capture method intended to be used by domain experts to represent the knowledge about how a particular system or process works. The IDEF3 method provides modes to represent both (1) Process Flow Description to capture the relationships between actions within the context of a specific scenario and (2) Object State Transition to capture the allowable transitions of an object in the domain. The IDEF3 tool will provide automated support for each mode of the method. The IDEF4 method is a graphically oriented methodology for the object oriented design of computer systems. The IDEF4 method provides a method for capturing the (1) Class Submodel or object hierarchy, (2) Method Submodel or the procedures associated with each classes of objects, and (3) the Dispath Matching or the relationships between the objects and methods in the object oriented design. The IDEF4 tool will provide automated support for the IDEF4 method.

The requirements specified in this document describe the capabilities that a fully functional IDEF3 or IDEF4 automated tool should support. However, it is impossible for the prototype tools to be developed under this contract to exhibit these capabilities completely. Instead, the prototypes will attempt to address the major areas of functionality so that the experience with those areas that will prove most difficult can be gained and can be documented.

The requirements for the automated IDEF3 description capture tool are found in Section 3.0. The purpose of this section is to:

(1) Document the basic structures of the IDEF3 tool including the Unit of Behavior (UOB), Links, Junctions, Referents, and Elaborations of the Process Flow Description and the Object States and Arcs of the Object Station Transition.

(2) Define the commands of the tool associated with each structure within each mode of the IDEF3 modeling method. These commands include (as a

minimum) commands for creating, editing, deleting, and copying these structures.

(3) Discuss organization of the IDEF3 descriptions. This functionality will allow the user to access the information about the scenario and decompositions of the models.

(4) Provide the requirements for the user interface that will support (1) browsing, (2) grouping or clustering, and (3) report generation features of the IDEF3 tool.

(5) Provide the requirements for the IDEF3 integration to other modeling tools, simulation modelers, and database managers.

(6) Provide the requirements for the model management requirements of the IDEF3 tool that will allow the user to save and load the complete model with the associated elaborations and decompositions.

The requirements for the automated IDEF4 design tool are defined in Section 4.0. The purpose of this section is to:

(1) Document the basic structures of the IDEF4 tool including Classes, Class Invariant Data Sheets, Features, Inheritance Links, Type Links, Method Sets, and Contract Data Sheets.

(2) Define the commands of the IDEF4 tool associated with each structure. The commands include creating, editing, copying, and deleting each of the various structures.

(3) Document the various ``view'' diagrams included in the IDEF4 tool including Inheritance Diagrams, Type Diagrams, Protocol Diagrams, Method Taxonomy Diagrams, and Client Diagrams.

(4) Provide the requirements for the IDEF4 tool user interface. This user interface is centered around the various views associated with the design model being developed. As a result, the IDEF4 tool must

include a powerful command set to provide effectively the automated support for a modeler. The user interface includes the functionality to select, find, and search through the various view diagrams. In addition, these requirements include the clustering or grouping operations and report generation capabilities.

(5) Provide the requirements for the effective integration of the IDEF4 model with other modeling tools and utilities.

(6) Provide the requirements for the model management requirements of the IDEF4 tool that will allow the user to save and load the complete model with the associated model kits.

Also, the Appendices of this Requirements document contain IDEFØ and IDEF1 models of the IDEF3 Process Flow Description Methodology and the IDEF4 Object Oriented Design Method. The IDEFØ and IDEF1 modeling methodologies provide effective means of performing functional decomposition and information modeling, respectively. These models were developed to illustrate the functionality and information that the automated tools must support. Specifically, Appendices A and B contain the IDEFØ model and the IDEF1 model, respectively, of the IDEF3 methodology, while Appendices C and D contain the IDEFØ model and the IDEF1 model, respectively, of IDEF4.

2.0 Knowledged Based Meta-Requirements

To build an effective tool requires that as much knowledge about the process being automated be encoded into the tool. A modeling tool with no understanding of the modeling methodology is usually no more than a specialized drafting system. With an understanding of the methodology present in the system, the tool can speed up the modeling process by automatically validating models to ensure that the model adheres to the rules and conventions of the methodology and by automatically updating the model to reflect changes made by the modeler. For example, if two objects in a model were related by a link in some methodology and one of the objects were deleted, a "smart" tool would want to also delete the link that related the two objects. Though this is a simple example, it reflects the kind of things that a knowledge based tool should do to increase the productivity of the modeler.

Some of the capabilities required in an knowledge based tool are listed below:

Anticipate Modeler Intentions

A tool should have an understanding of the operations to be performed by the user. This would provide the ability to anticipate the steps to be carried out in performing that operation and allow the tool to automatically execute those steps.

Revise Models Intelligently

This requirement coincides with the example given above. Whenever an operation results in a change to a model, revise all structures that are somehow related to the revision so that the changes are automatically reflected in the entire model. This could be coined as the "ripple effect" where the system should recognize and smooth all ripples created by a change to a model.

Incorporate Conflict Resolution Strategy

When an operation performed on a model results in a conflict, the tool should have the ability to resolve the conflict on its own, instead of querying the user for the appropiate action to take.

Maintain Individual Model Consistency

After every operation that would have an effect on the model, but before the operation has been committed to the model, the effect of the operation should be analyzed to determine if the changes are valid. It is only after the validity has been determined that the changes should be reflected in model.

Maintain Merged Models Consistency

This requirement deals with the situation where two or more valid models might be merged to produce an invalid or inconsistent model. The tool should have the ability to recognize that the merge will result in inconsistencies and should have built-in strategies to resolve those conflicts so that the resultant model will be as consistent as possible.

Support Generation of Other Models

The database structure of the tool should have the ability to support the translation of the information represented in a description or design model into other formats. For example, the tool might translate the process description into a SIMAN simulation model. The code generated by the tool could then be run to analyze the process description. This ability requires that the tool have an understanding of the semantics of the model so that a effective translation can occur.

Though the IDEF3 and IDEF4 requirements that follow in the next two sections may not explicitly refer to these knowledge based requirements, it is important that every operation support and abide by the previous requirements. As a result, these knowledge based requirements can be considered to be the modeling tool meta-requirements.

3.0 IDEF3 Requirements

IDEF3 is a scenario driven process flow description capture method. Its goal is to provide a structured method for expression of the domain experts knowledge about how a particular system or organization works. To automate effectively the IDEF3 description process, it is imperative that a thorough understanding of the methodology be reflected in the tool. The sections of this paper outline and describe the functions and capabilities necessary for a tool to support effectively the IDEF3 modeling methodology.

Two modeling modes exist within IDEF3: process flow description and object state transition description. A process flow description indicates "how things work" in an organization while an object state transition description summarizes the allowable transitions an object may undergo throughout a particular process. The IDEF3 tool will support both modes of information capture and will provide the necessary functionality to integrate properly the various process descriptions and object state transition descriptions that may be developed to model an enterprise.

3.1 IDEF3 Entity Requirements

Both the Process Flow Description and Object State Transition Description contain units of information that make up the description. These entities, as we call them, are the basic units of an IDEF3 model. Because of their atomic nature, these entities require operations to be performed on them. This section describes the operations that are required for the manipulation of these entities.

3.1.1 Process Flow Entity Requirements

An IDEF3 Process Flow Description captures a network of relations between actions within the context of a specific scenario (see Section 3.2.1). The intent of this description is to show how things work in a particular organization. Figure 1 presents an example IDEF3 Process Flow Diagram.

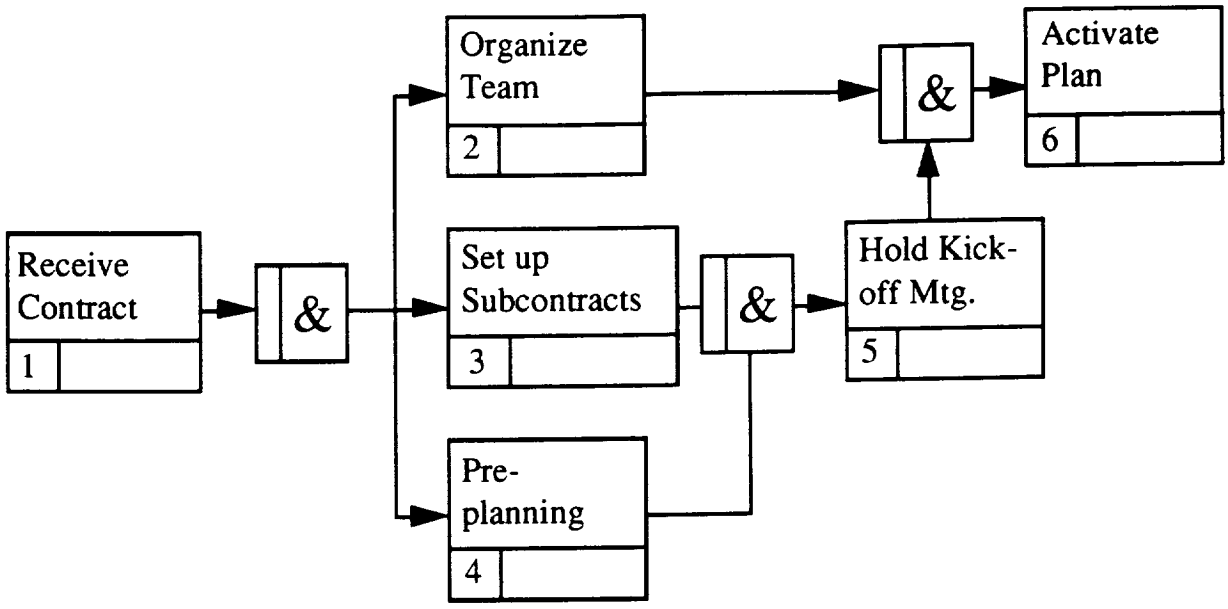


Figure 1. An Example Process Flow Diagram

An IDEF3 Process Flow Diagram consists of the following structures:

Units of Behavior (UOBs),
Junctions,
Links,
Referents, and
Elaborations.

The development of an IDEF3 Process Flow Diagram will consist of the generation and manipulation of these model entities. The automated IDEF3 tool should make this task as simple and as intuitive as possible.

3.1.1.1 Unit of Behavior (UOB)

The Unit of Behavior (UOB) is the basic unit of the Process Flow Diagram and is used to represent complex states of affairs. In a diagram, a UOB is displayed with its label, node number, and optional IDEFØ activity reference number. Figure 2 shows how a UOB would appear in a diagram. The label should be verb-based and provide some indication as to what process is being represented by that particular UOB. Though not displayed in the diagram, each UOB also has associated with it a name that must be unique across the process flow and a textual glossary entry to provide additional information about the process being represented. Also, a UOB can have

decompositions (Section 3.2.2) and elaborations (Section 3.1.1.5) associated with it.

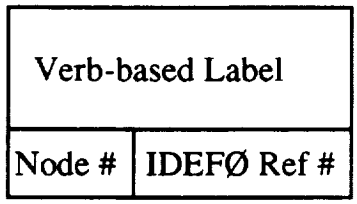


Figure 2. IDEF3 Unit of Behavior

The Node # is assigned to a UOB sequentially in the order in which the UOBs are created. But, when the process model is released, the nodes are renumbered according to the following rules:

1) The UOBs on the top level scenario are numbered sequentially from left to right and from top to bottom.
2) The UOBs on any subsequent diagram are numbered in the same fashion, except that a prefix is attached with a period to the number. The prefix is composed of the number of the UOB linked to that scenario followed by a period and then the letter "o" if the diagram represents an objective view decomposition of the UOB linked to that scenario. If the scenario represents a role view, then use a "v" instead of an "o".

The following is a list of minimal operations that must be supported by an automated IDEF3 tool to manipulate effectively Units of Behavior. The description of each function gives an idea of the things that must be done to achieve the desired functionality.

3.1.1.1.1 Create Unit of Behavior

At the time of creation of a Unit of Behavior, the user will be allowed to input the name of the UOB, the Label of the UOB, and any Text associated with that UOB. In addition, the user can specify an IDEF0 activity number to provide a cross reference with an associated activity model. When the user has completed inputting this information, the UOB is created, added to the model, and appears in the current Process Flow Description.

3.1.1.1.2 Edit Unit of Behavior

At any time after creation of a Unit of Behavior, the UOB can be edited. With this operation, the user will be allowed to modify the Name, Label, Description, or IDEF0 activity associated with a Unit of Behavior. When this operation is terminated, any changes made will be reflected in the model and on the terminal screen.

3.1.1.1.3 Copy Unit of Behavior

The tool should allow form Units of Behavior to be copied. Implicit copying will allow the same Unit of Behavior to be referenced in the same description again or in several different scenarios and decompositions at the same time. But the system should also provide for the explicit copying of a UOB. In this case, the structure of the UOB will be copied, but no information will be stored in the new copy. This allows the user to construct quickly UOBs that have very similar structures (including elaboration and decompositions) but that store different information in those structures.

3.1.1.1.4 Delete Unit of Behavior

At any time after creation of a Unit of Behavior, the UOB can be deleted. When the modeler chooses to perform this operation, the tool will delete the Unit of Behavior from the model. In addition, any ripples resulting from this deletion will be smoothed as well. This would include things like removing links that were attached to this UOB, removing decompositions of the UOB from the model, and checking other decompositions that contain this UOB to see if the UOB should be deleted from that decomposition.

3.1.1.1.5 Unit of Behavior Coercion

Often, it may become evident that a Unit of Behavior has evolved into its own scenario (see Section 3.2.1). As a result, the tool should support the coercion of a Unit of Behavior into a scenario in the current model. The name of the UOB will become the name of the scenario. The dual of this operation is specified in Section 3.2.1.5.

3.1.1.2 Link Requirements

In IDEF3, links are used to denote distinguished relations between UOBs. Figure 3 shows the different types of links supported in a process flow diagram. The simplest link is the solid-lined Precedence Link. The only thing represented by a precedence link is the simple temporal precedence between the instances of one UOB type and those of another UOB type. In other words, each instance of a UOB at the front of the link must complete before the corresponding instance of the UOB at the destination of the link begins. The dashed Relational Links carry no pre-defined semantics. The merely highlight the existence of a relationship between two or more UOBs. The semantics of the relationship would be captured in the glossary entry associated with the link. Finally, the Object Flow Links highlight the participation of an object in two UOB instances. The Object Flow Link carries the same temporal semantics as the Precedence Link.

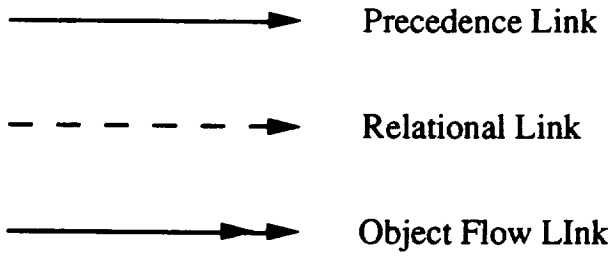


Figure 3. IDEF3 Link Types

Links may start or terminate at any point on a UOB or Junction. However, by convention, process flow diagrams are laid out so that the flow of objects and temporal precedence is from left to right and top to bottom.

3.1.1.2.1 Create Link

The tool will support the creation of Links. At the time of creation, the user will be allowed to specify the type (precedence, relational, or object flow) of link to be used and the Units of Behavior or Junctions to be related by the link. If a link is to be created between a Junction and several UOBs, the tool will allow the user to input the multiple UOBs at one time so that the number of Link

creations will be reduced. Also, at the time of creation of a relational link, the user may optionally provide a link description to detail the semantics of the relational link being created.

3.1.1.2.2 Edit Link

At any time after creation of a link, the link may be edited. This may involve changing the UOBs that are related by the link, changing the type of the link, or editing the description of a relational link. Once the changes are completed, the tool should analyze the effect that the changes would have on the model. If the change would result in an invalid model (for example, if a precedence link related a UOB to itself), the user should be notified that the changes are invalid and that they are being ignored. Otherwise, the changes are reflected in the current model.

3.1.1.2.3 Copy Link

The tool should allow Links to be copied. When this operation is specified for a certain link, the tool should create a new Link in the diagram and copy the information maintained in the original link into the new link. The only requirement is that the modeler change at least one of the UOBs related by the link. Otherwise, multiple links specifying the same relationship will exist within the model.

3.1.1.2.4 Delete Link

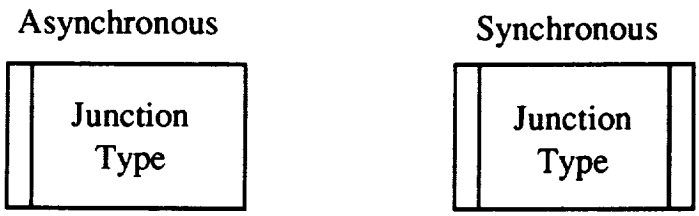
At any time after creation of a link, the link may be deleted. When this occurs, the link is simply removed from the current model.

3.1.1.3 Junction Requirements

An IDEF3 Junction is used to highlight special types of constraints on the possible sequencing relations among UOBs. The junctions link branches of the process flow that can proceed independently of each other. Each junction has an associated type and sequencing interpretation. The types supported in IDEF3 are AND, OR, and XOR. The semantics of these types are equivalent to their logical meanings. The sequencing interpretation can be either (1) synchronous which means the desired effect must occur at the same time or (2)

asynchronous which means the desired effect can occur in any sequence.

Different interpretations also exist for junctions initiating a branch and junctions terminating a branch. A junction initiating a branch specifies a constraint on the instantiation of the following processes while a branch terminating a branch constrains the termination of the processes preceding the junction. As such, it is possible for invalid junction combinations to be specified. For example, an XOR initiating a branch that is terminated by an asynchronous (or synchronous) AND junction is invalid. Since the XOR specifies that that only one of the following processes can be realized, the ending AND condition can never be satisfied. The IDEF3 tool should recognize these pathological situations and indicate them to the user so that appropriate actions may be taken.



Junction Type
& AND
O OR
X XOR

Figure 4. IDEF3 Junctions

Figure 4 shows how Junctions would appear in the process flow diagram. The asynchronous junction is represented by a box with a line running down the left side. The synchronous junction is the same as the asynchronous except that an additional line runs down the right side of the box as well as the left. The type of the junction is specified by placing the type symbol (&, O, or X) in the center of the junction box. Note that Exclusive Or (X) requires no type of synchronization since only one of the UOBs following the junction can be instantiated. However, by default, an XOR junction uses the Asynchronous box in the process diagram.

Here is the list of required operations for IDEF3 Junctions.

3.1.1.3.1 Create Junction

The tool will support the creation of Junctions. At the time of creation, the user must specify the interpretation (asynchronous or synchronous) and the type (AND, OR, or XOR) of the junction. In addition, the user must specify if the junction will be used for fan-out or fan-in purposes. When the operation is completed, the junction will be added to the current model.

.c.3.1.1.3.2 Add Link to Junction

After a junction has been created, it may be necessary to add another degree to the amount of fan-in or fan-out for the particular junction. This operation would provide the ability to add links to the junction.

.c.3.1.1.3.3 Remove Link from Junction

After a junction has been created, it may be necessary to add another degree to the amount of fan-in or fan-out for the particular junction. This operation would provide the ability to add links to the junction.

3.1.1.3.4 Edit Junction Definition

At any time after creation, the user can edit a Junction. With this operation, the user will be allowed to modify the junctions interpretation, type, or fan in/out purpose. When the user completes modification of the junction, the tool will determine if the changes cause any conflicts in the current model. If there are problems, the user will be notified and the changes ignored. Otherwise, the changes will be reflected in the current model.

3.1.1.3.5 Copy Junction

The tool should allow the user to copy a Junction. When this operation is performed, a new Junction object should be created with the same characteristics as the original Junction.

3.1.1.3.6 Delete Junction

At any time after creation, the user can delete a Junction. When this occurs, the junction is removed from the current

model. In addition, any links that may be associated with the deleted junction (see below) are also deleted.

3.1.1.4 Reference Requirements

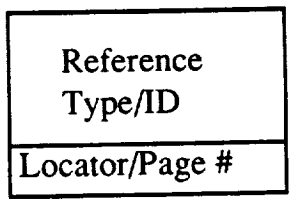


Figure 5. IDEF3 Reference

A Reference allows modelers to:

- Span multiple pages in a diagram layout.
- Refer to a previously defined UOB without duplication of its definition.
- Emphasize the description of particular objects or relations in the elaboration.
- Tie in specific examples of referenced data or object.
- Associate special constraint sets to junctions.
- Form references of links to Object State Transition descriptions.

Figure 5 shows how references will appear in the diagrams. A Reference is represented by a box with a horizontal line near the bottom. The type of information being referenced or a reference identifier (if this reference is not defined elsewhere) is displayed in the center of the box. Below the horizontal link, the link to the information to be referenced is specified.

3.1.1.4.1 Create Reference

The tool will support the creation of References. At the time of creation, the user will be prompted for the type of information to be represented by the reference. When completed, the reference will be added to the current model.

3.1.1.4.2 Edit Reference

At any time after creation of a reference, the reference may be edited by the user. If the changes to the reference are valid, those changes will be reflected in the current model.

3.1.1.4.3 Copy Reference

The tool should allow references to be copied. When this operation is specified for a certain reference, a new reference is created and added to the model. In addition, any information that is stored in the original reference is also copied to the new referent.

3.1.1.4.4 Delete Reference

At any time after creation of a reference, the reference may be deleted. When this action is taken, the selected reference is simply removed from the current model.

3.1.1.5 Elaboration Requirements

An Elaboration is a description of a Unit of Behavior in terms of objects and constraints that exist on those objects during the unit of behavior. IDEF3 allows for three levels of elaboration specification to be used by the area expert, the analyst modeler, and the software systems developer, respectively. At the first specification level, the elaboration is expected to be captured on an elaboration form. This form captures the information from the area expert in natural language textual descriptions and presents this information in a structured display that includes an object list, a fact list, and a constraint list.

The elaboration also provides a classification structure for the participating objects. Each object can be:

- tagged as an "agent" if that object is considered to be the effector of the UOB;
- tagged as "affected" if the relations to that object are created or changed by/during the UOB;
- tagged as a "participant" if no causality or transformation is associated with that object as a part of the UOB description;
- tagged as "created" by the UOB;
- tagged as "destroyed" by the UOB.

These classifications are optional, but do allow for automated analysis against the format semantics of IDEF3.

3.1.1.5.1 Create Elaboration

The tool should support the creation of Elaborations. An Elaboration is attached to a UOB to describe objects that exist within the process captured by that UOB and any relations that may exist between these objects. When this operation is performed, the user will identify the objects that will be part of the elaboration and any relations between these objects. The user must then indicate which UOB this elaboration is to be attached to. When this is done, the Elaboration is added to the current model and the UOB is modified to indicate that it now has an Elaboration.

3.1.1.5.2 Edit Elaboration

At any time after creation of an Elaboration, the Elaboration may be edited by the user. In this manner, the user may add, edit, or delete objects and relations from the Elaboration. When completed, the model will be updated to reflect the changes.

3.1.1.5.3 Copy Elaboration

The tool should allow Elaborations to be copied. When this operation is specified for a certain Elaboration, a new Elaboration is created and added to the model. Any information that is maintained in the original Elaboration is then copied into the new Elaboration.

3.1.1.5.4 Delete Elaboration

At any time after creation of an Elaboration, the Elaboration may be deleted by the user. When this occurs, the Elaboration is removed from the model and the UOB to which the Elaboration was attached is updated to reflect the removal of the elaboration.

3.1.2 Object State Transition Entity Requirements

An Object State Transition Diagram is used to capture an object centered view of a process. This view cuts across the process diagrams and summarizes the allowable transitions of an object in the domain. The entities of an Object State Transition Description are:

1) Object States
2) State Transition Arcs
3) Process Description Network References

The IDEF3 tool will allow various operations to be performed on these entities.

3.1.2.1 Object State Requirements

An Object State is defined in terms of attributes. These attributes may be defined in an IDEF1 model and cross referenced in the Object State Transition Diagram. An Object State can also have Pre-transition and Post-transition requirements associated with it. These requirements specify the conditions that (1) must be met before a transition can begin for pre-transition requirements or (2) must be complete for post-transition requirements. The requirements are specified by a list of attributes/value pairs. The values of the attributes must match the specified values for the requirements to be met.

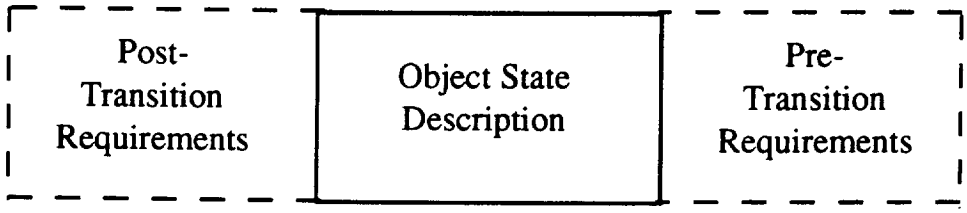


Figure 6. IDEF3 Object State Description

Figure 6 shows how an object state description would appear in a Transition diagram. The solid box represents the description of the actual state. The dashed boxes will include the constraints that must be met before an object state can begin (pre-transition) or end (post-transition) a state transition.

3.1.2.1.1 Create Object State

The tool will support the creation of Object States. At the time of creation, the user will be asked to provide the Name, Label, and a Description of the Object State. In addition to this descriptive information, the user may also specify Pre-transition or Post-transition Restrictions for

the Object State during the creation of that Object State. When this information has been input, the Object State will be created and inserted into the current Transition Diagram.

3.1.2.1.2 Edit Object State

At any time after creation of an Object State, the Object State may be edited by the user. This may involve editing the descriptive information of the Object State or adding, deleting, or editing restrictions in the Pre-transition or Post-transition restriction list of the Object State.

3.1.2.1.3 Copy Object State

The tool should allow Object States to be copied. When this operation is specified for a certain object state, a new object state is created and added to the model. Then, the information stored in the original object state should be copied into the new object state.

3.1.2.1.4 Delete Object State

At any time after the creation of an Object State, the Object State may be deleted by the user. When this action is taken, the Object State is deleted from the current diagram. If arcs have been defined that link this Object State to other States, those arcs are automatically removed from the current diagram as well. In addition, if the Object State to be deleted is used in other State Transition diagrams, the user will be prompted to determine if the deletion should occur in the other diagrams as well.

3.1.2.2 Transition Arc Requirements

In an Object State Transition Diagram, an Arc simply specifies a transition from one object state to another. Each arc has a label and a glossary description attached to it. In addition, a Reference can be attached to the arc to specify the execution of an IDEF3 process description before the transition can be executed.

3.1.2.2.1 Create State Transition Arc

The tool will support the creation of Arcs. At the time of creation, the user will be asked to provide the Name, Label, and a Description of the Arc to be created as well as indicate the two Object States that will be linked by this

arc. In addition, the user must specify the Object State Transition Diagram or Process Flow Diagram that will control the transition between the two Object States to be linked. Once this information has been provided, the arc will be created and added to the current diagram.

3.1.2.2.2 Edit State Transition Arc

At any time after the creation of an Arc, the Arc may be edited. This may involve editing the Name, Label, or Description of the Arc or changing the Object States that are linked by the Arc. With this operation, the user may also change the Object State Diagram or Process Flow Diagram that is attached to the Arc. When the editing is completed, the current diagram is updated to reflect the changes.

3.1.2.2.3 Copy State Transition Arc

The tool should allow Arcs to be copied. When this operation is specified for a certain arc, a new arc should be created and added to the model. Any information represented by the arc would also be attached to the new arc. However, the user would have to specify the new object states to be related by this new arc.

3.1.2.2.4 Delete State Transition Arc

At any time after the creation of an Arc, the Arc may be deleted. When this action is taken, the Arc to be deleted would be removed from the current diagram. In addition, any reference that the arc made to other diagrams or to Object States would be removed as well.

3.1.2.3 Process Description Network References Requirements

A Process Description Network Reference provides a link between an object state transition diagram and a process description. It may be required that the referenced process complete before the transition can occur.

3.1.2.3.1 Create Process Description Network Reference

The tool should allow the user to create a network reference. The user would have to provide the UOB name

or scenario name to indicate exactly which network should be referenced.

3.1.2.3.2 Edit Process Description Network Reference

Any time after a network reference has been created, it can be edited. This would involve changing the UOB or scenario network referred to by the reference.

3.1.2.3.3 Copy Process Description Network Reference

The tool should allow network references to be copied. In this situation, an additional reference to the network in the original reference will be created and the information copied to the new reference.

3.1.2.3.4 Delete Process Description Network Reference

Any time after a network reference has been created, that reference can be deleted. When this occurs, the reference to the network is removed from the transition diagram and the diagram is updated.

3.2 IDEF3 Organization Requirements

In addition to the Entity Requirements, the IDEF3 tool will provide the functionality to organize the entity elements into complex IDEF3 structures. The types of organization supported include:

1) Scenarios
2) Decomposition Networks
3) Object Taxonomies
4) Relational Networks
5) Scenario Correspondence

It should be noted that these structures are not the only means of organization in an IDEF3 model. In fact, the creation of a Link or Junction in a Process Flow Description is in itself an organizational operation, but the operations described in this section represent higher levels of organization that require more complex operations to ensure that they are executed correctly.

The Scenario is the basic organizing structure for IDEF3 process flow descriptions, and an IDEF3 model will normally contain a number of different scenarios. A Decomposition is a breakdown of a specific

UOB in terms of other UOBs and their associated links. In actuality, a Scenario is not much different from a Decomposition, in that both capture process descriptions through the definitions of UOBs and their relationships. The main difference comes from their context. A Scenario is a subunit of the entire description while a Decomposition is associated with a UOB. Despite the similar structure of the Scenario and Decomposition, their difference in context requires operations on these two structures to be separated.

3.2.1 IDEF3 Scenario Requirements

A Scenario defines a context in which to consider a particular IDEF3 description. It is reasonable to assume that the same process can be viewed from several different perspectives. By supporting scenarios, we allow these different perspectives to be represented in the same model instead of requiring that each view have its own model.

3.2.1.1 Create Scenario

The tool will support the creation of Scenarios. With this operation, the modeler will specify the name of the Scenario to be created as well as any glossary information or textual descriptions to be associated with the scenario. When he has done that, the system will create all the internal structures required to represent a Scenario. When this has been done, the system will be ready to accept the process description.

3.2.1.2 Edit Scenario

The purpose of this operation is to support multiple scenarios within an IDEF3 model. In providing this capability, we assume that the user may be making modifications to one Scenario and then decide to make changes to a completely different Scenario. When this occurs and the user specifies the name of the Scenario to edit, the system "swaps" out the old Scenario and the new Scenario. When this is completed, modification to the loaded Scenario, including changes to the glossary, text descriptions, and process descriptions, can occur.

3.2.1.3 Copy Scenario

The tool should allow an entire scenario to be copied. When this operation is specified, the entire structure of the

scenario would be duplicated and a new scenario created and added to the model.

3.2.1.4 Delete Scenario

At any time after a Scenario has been created, the Scenario can be deleted. If this action is taken, the current Scenario will be deleted from the IDEF3 description. Any objects in this Scenario that are referenced in other Scenarios will still remain in the other Scenarios. This deletion will have no effect on their use in other parts of the model.

3.2.1.5 Scenario Coercion

Often, a scenario that exists in an IDEF3 model may need to be referenced in another scenario of the same model. To support this, the tool should provide the ability to coerce a scenario into a UOB that can be manipulated by other scenarios.

3.2.2 IDEF3 Decomposition Network Requirements

A Decomposition is just a more detailed description of a UOB in terms of other UOBs and their relations. Two types of decompositions exist within IDEF3: Objective View Decompositions and (Role) View Decompositions. Any UOB can have one Objective View and many View decompositions. The distinguishing feature between these two types of decompositions is their use of objects. In an Objective View, all objects specified in the Elaboration for a UOB must be represented in the decomposition. As such, an Objective View of a UOB cannot exist unless the Elaboration for that UOB has already been defined. A View decomposition is a decomposition that does not satisfy that requirement.

Figure 7 shows the relationship between a UOB and its decomposition. The decomposition is just an expansion of the UOB itself and allows the modeler to describe the processes represented by UOBs in differing levels of detail.

3.2.2.1 Create Decomposition

The tool will support the creation of Decompositions of UOBs. With this operation, the user will be able to describe a particular UOB in further detail. At the time of creation, the user must specify the type (objective view or view) of Decomposition to be created. Actually, this type is important only when an Elaboration for the UOB to be

decomposed has been specified, since an objective view decomposition requires that all objects specified in the Elaboration be accounted for in the decomposition. Once the type information has been specified, the UOB to be decomposed is updated to indicate that a new decomposition has been added to it and the tool environment is updated to allow the user to begin storing information in the decomposition.

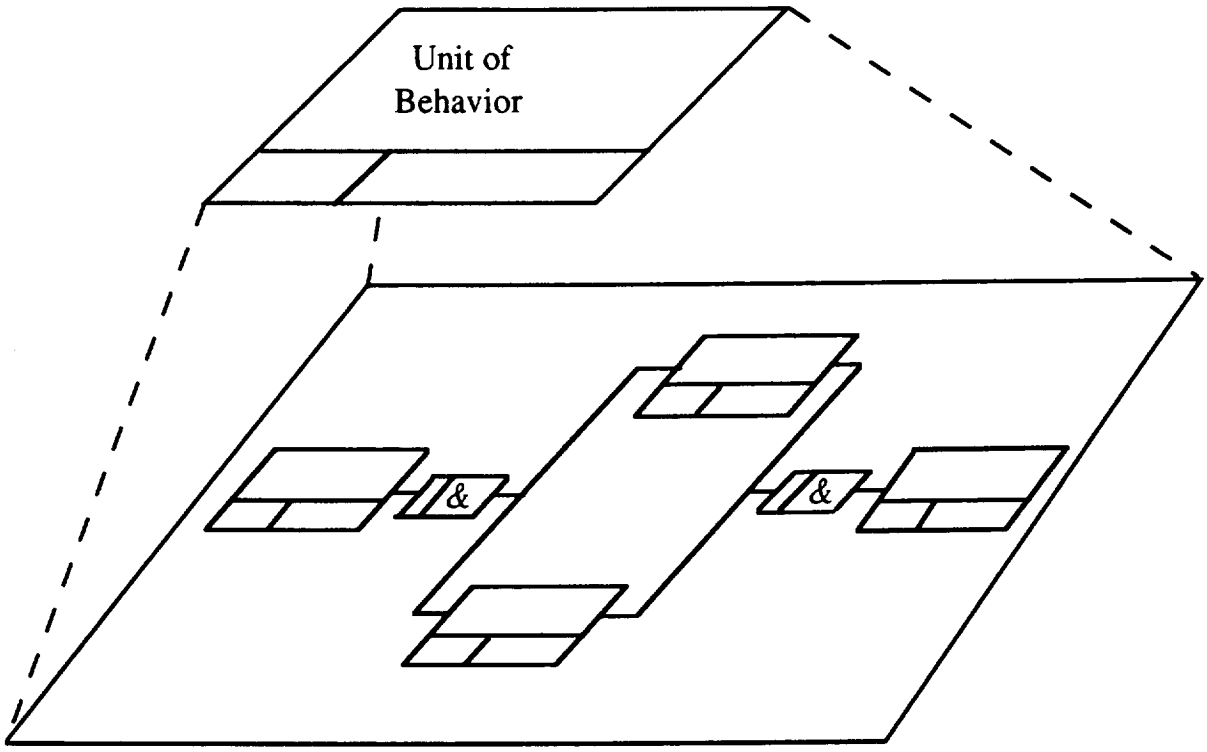


Figure 7. An IDEF3 Decomposition

3.2.2.2 Edit Decomposition

At any time after creation of a Decomposition, the Decomposition can be edited. This would involve adding, editing, or deleting UOBs, Links, or other structures to the Decomposition. This capability would be somewhat implicit in the tool since the Browsing functionality (see User Interface Requirements) would allow easy movement into and out of Decompositions. When a Decomposition is visible, changes to that Decomposition can be made.

3.2.2.3 Copy Decomposition

The tool should allow an entire decomposition to be copied. When this operation is specified, the entire structure of the decomposition is duplicated in another location. It should be noted that no new structures are created unless the user specifically requests them. Without new structures, any changes made to the copy will automatically be reflected in the original decomposition.

3.2.2.4 Delete Decomposition

At any time after creation of a Decomposition, the Decomposition can be deleted. When this action is taken, the parent UOB of the Decomposition is updated to reflect that this decomposition is no longer a part of that UOB. In addition, the structures used in the decomposition will be removed from the IDEF3 model, unless the objects are used or are referenced by other structures in the model.

3.2.3 IDEF3 Object Taxonomy Requirements

An automated IDEF3 tool should support the development of Object Taxonomies. These structures would define relationships between objects that exist in the various process flows and object state transition diagrams that make up an IDEF3 model. They are useful in capturing information that is not easily represented in a process flow. For example, two UOBs that have no relation (i.e., no links) to each other within the scope of a process flow may be related because the UOBs involve the same or similar objects. This relationship could be captured effectively in an object taxonomy.

3.2.4 IDEF3 Scenario Correspondence Requirements

An additional organization requirement of an automated IDEF3 tool deals with scenario correspondence of objects. The requirement here is that objects involved within a certain scenario should be indexed by that scenario. This would provide the ability to query a scenario about the objects involved in the scenario.

3.3 IDEF3 User Interface Requirements

Perhaps the most critical portion of a modeling tool is the user interface. The goal of any automated tool is to make the process

being automated easier to perform with the tool than without the tool. To meet this goal in the IDEF3 tool, it is important that certain capabilities be available to the user to promote easy movement and access to the various components of the diagrams.

3.3.1 Browsing Requirements

The most important aspect of the IDEF3 tool user interface is the display of the various diagrams being developed by the modeler and the information maintained in those diagrams.

3.3.1.1 Scenario Selection

The tool should allow the user to move easily between different scenarios in the current model. It should also be obvious which scenario is currently selected so that the users will not be confused.

3.3.1.2 Object State Transition Diagram Selection

The tool should allow the user to easily move between different Object State Transition Diagrams in the current model. It should also be obvious which transition diagram is currently selected so that the users will not be confused.

3.3.1.3 Find References

The tool should allow to user to find all references of a particular IDEF3 structure. This operation would return a list of all diagrams where the specified object is referenced.

3.3.1.4 Speed Search

Due to the size of IDEF3 process descriptions, the tool should be equipped with a Speed Search utility. This function would allow the user to input the name of a particular object that he wishes to examine, and the tool would redisplay the diagram to display the object whose name most closely matches the partial string entered by the user. This is equivalent to string searching in a text editor, except that the diagram is being searched instead of a block of text. This capability should apply to both process flow diagrams and object state transition diagrams.

The following functions deal with movement through the process flow diagram hierarchy.

3.3.1.5 Decomposition Selection

The tool should allow the user to move down the hierarchy of process diagrams. After the user indicates that he wishes to move to the decomposition of a specific UOB, the tool should display the diagram of the desired decomposition.

3.3.1.6 Parent Selection

This function would be the inverse operation of the Decomposition Selection. After the user has viewed a Decomposition, he should be able to easily move back up the hierarchy to view the diagram in which the parent UOB is used.

The following capabilities represent alternative viewing modes of the IDEF3 diagrams that should be supported by the tool. Instead of viewing the IDEF3 diagrams graphically, these modes may display information in a list of textual information. Each of these modes should have operations similar to those described previously in this section to allow for rapid browsing of the information presented.

3.3.1.7 Decomposition Listing

In this mode, the tool will arrange all the UOBs in the current view into a tree hierarchy and present this tree as an indented list. The tree will represent the hierarchy of the decompositions in the diagram. The top level UOBs will be listed normally while UOBs that appear in decompositions will be indented appropriately under the UOB in whose decomposition the indented UOB occurs. This mode would be useful to determine where certain UOBs occur in the current process diagram, but when temporal information is not necessarily important.

3.3.1.8 Elaboration Listing

This mode would perform a similar function as the Decomposition Listing except that the Elaborations would be indented under the UOBs instead of the Decompositions. This information would be useful to determine which UOBs manipulate certain objects in the environment being modeled.

To facilitate the rapid development of these IDEF3 models, it would also be useful if the tool could easily move between these different

display modes. For example, in the Decomposition listing, once the user has identified a particular decomposition, the tool should be capable of displaying the process flow diagram for that decomposition at the request of the user. As the IDEF3 models grow in size, this type of functionality will allow for rapid searching and movement within the model and thus provide for rapid development and modification of the models.

3.3.2 Group Operation Requirements

To reduce the number of repeated operations, and thus to speed up the development of IDEF3 models, it would be useful if certain operations could be performed on groups of objects in the diagrams.

3.3.2.1 Copy Structure by Group

The tool should allow the user to select several objects from a diagram to be copied as a single unit. By copying the structure, the user is saying that he wants to copy how the various objects in the unit are arranged, not the information that is stored in the objects. For example, a diagram may contain a UOB followed by an asynchronous AND junction that fans out to three UOBs. At another point in the diagram, the same structure may be required. With this ability, the user can simply copy the structure and then input the new information. It should be noted that when this operation is performed, new objects are created and added to the diagram. It is then up to the user to input the appropriate information into the objects.

3.3.2.2 Copy Information by Group

The tool should allow the user to select several objects from a diagram to be copied as a single unit. In this mode, the user is copying the information stored in the structure as well as the structure of the objects. As a result, no new objects are created. The same objects are used in the new location as well as in the original unit.

3.3.2.3 Delete by Group

The tool should allow the user to delete several objects from a diagram as a single unit. After executing this operation, the selected objects will be removed from the current diagram. In addition, the diagram will be checked to see if any loose ends remain from the group deletion. If

so, these conditions should be appropriately handled by the tool.

3.3.3 Activation Visualization Requirement

It is also important that the automated IDEF3 tool provide a means to visually view an activation of a process flow or object state transition. An activation in this sense is roughly equivalent to a simulation of the flow or transition description and provides modeler to analyze his models effectively without having to use a sophisticated simulation model. When the modeler requests an activation, an instantiation of the process flow or state transition should be initialized and visual cues should be available to indicate the status of the process flow activation.

3.3.4 Report Generation Requirements

To assist in the development of reports surrounding an IDEF3 model, it is important that the following functions be supported.

3.3.4.1 Hardcopy

The tool should provide the ability to generate printouts of the process flow and object state transition diagrams.

3.3.4.2 ASCII File Dump

The tool should provide the ability to generate an ASCII file of the information maintained in and IDEF3 model. This would enable database applications to be generated for the information maintained in the IDEF3 models.

3.3.3.3 Note Attachment

The tool should support the attachment of notes to description elements so that information about the structure can be captured. These notes would be organized in a HyperText fashion to allow the greates degree of flexibility.

3.4 IDEF3 Integration Requirements

It is also important that the IDEF3 tool address integration with other modeling methodologies. This is especially true since a Unit of Behavior can directly reference an IDEFØ activity and an Object State

can directly reference IDEF1 attributes. It is possible to build totally integrated tools for each of the methodologies, but practical limitations limit this possibility (IDEF modeling is usually performed by a team of modelers on several machines, requiring a networked modeling environment with a sophisticated file server to control access to the models). Therefore, without a common database for the different methodologies, it is important that the tool provide some mechanism for loading information from the various models into the IDEF3 modeling environment.

3.4.1 IDEFØ Integration

In IDEF3, a UOB can directly reference an IDEFØ activity. As a result, the tool should provide the ability to load in activity and concept information from an IDEFØ model.

3.4.2 IDEF1 Integration

In IDEF3, an object described in a UOB's elaboration or in and Object State Transition Diagram are often derived from entity classes maintained in an IDEF1 model. Also, the attributes defined for an entity class can be used as part of the pre- and post-transition requirements of an object state. For these reasons, the tool should provide access to information stored in IDEF1 models.

3.4.3 Consistency Maintenance

In addition to accessing the various models, it is important to ensure consistency in the IDEFØ, IDEF1, and IDEF1X models loaded into the IDEF3 tool. If any changes are made to the objects stored in these models, the tool should provide the capability to download these changes so that the original models can be updated accordingly.

3.4.4 Data Extraction

The tool should support the ability to extract information from an IDEF3 process description for use in other unspecified applications. This function would take advantage of the database operations described in Section 3.5.4 to access the information the modeler requires.

3.4.5 Easy Access to Simulation Modelers

The tool should also be designed to provide easy access to process description and object state transition data for other modeling utilities, such as Simulation Modelers. This

is a specialized case of the Data Extraction capability discussed above. This would allow the process captured in an IDEF3 model to be used as part of a simulation analysis process.

3.5 IDEF3 Information Management Requirements

Related to the development of models is the problem of model validation. Any automated tool should provide the means to ensure that models being created actually conform to rules and guidelines of the methodology. As a result, the tool should have the following capability.

3.5.1 Model Verification

The tool should have a working understanding of the IDEF3 Metamodel, a model of IDEF3 in IDEF1. This knowledge is imperative if the tool is to be effective. Without this information, the user would have free reign to develop any type of construct possible with the entities of IDEF3. By encoding the metamodel into the IDEF3 tool, the development of valid IDEF3 models is ensured by allowing only valid IDEF3 constructs to be created.

One of the most basic requirements, as well as the most important, is the ability to save and reload the information stored in a model. As a result, it is imperative that the IDEF3 tool provide these capabilities.

3.5.2 Model Saving

The tool should allow a model to be saved. At the completion of a modeling session, the user should be able to save the entire model. This would include saving all scenarios and all object state transition diagrams.

3.5.3 Model Loading

The tool should allow a model to be loaded. If models have been saved, the user should be allowed to load any of those models into the tool for browsing or editing purposes. More sophisticated tools could support access control so that only authorized users would be capable of loading a model in an edit mode. All other users would only be allowed to browse the model.

3.5.4 Model Database

The tool should support database-like operations to provide for the rapid extraction of information from the process model. The operations should allow the user to query against the model for information related to certain objects represented in the model.

The following two capabilities allow for a submodel to be used in other models. These utilities should allow a portion of one model to be written to a file and then later reloaded, as a submodel, into another model. This capability will greatly reduce the amount of work that must be duplicated.

3.5.5 Kit Save

The tool should allow a portion of model (kit) to be saved to a file or database. The model portion could be a specific scenario, decomposition, or object transition state diagram.

3.5.6 Kit Load

The tool should allow a kit to be loaded into the current model. The user should specify where the kit is to be loaded. If the type of the kit (scenario, decomposition, or object state transition diagram) agrees with the type of the location where the kit is to be loaded, the kit will be loaded into the current model.

3.6 Summary

This section has presented the minimal functionality necessary for the development of an effective tool for automating the IDEF3 modeling methodology. Four major areas of functionality have been addressed: IDEF3 Entity Requirements, IDEF3 Organizational Requirements, IDEF3 Integration Requirements, and IDEF3 Information Management Requirements. Each of these areas represent a major segment of an automated IDEF3 tool's functionality and should be treated as equally important when developing an automated tool.

4.0 IDEF4 Requirements

IDEF4 is a graphically oriented methodology for the object-oriented design of computer programs. It provides the necessary facilities to support the object-oriented design decision making process. Conceptually, an IDEF4 design model consists of two submodels: the Class Submodel and the Method Submodel. These two submodels are linked through the Dispatch Mapping, and together these three structures capture all the information represented in a design model. However, due to the size of the Class and Method Submodels, the designer never sees these structures. Instead, the designer makes use of a collection of smaller diagrams that effectively capture the information represented in the Class and Method Submodels.

The purpose of the IDEF4 tool is to make the creation, editing, and viewing of these view diagrams as simple as possible. The tool must provide the functions necessary to do this, but do so in a way that is intuitively obvious and simple to use. If the tool does not meet this one requirement, it will be useless since the tool will be more difficult to use than developing the design model by hand. The following sections outline the operations and functions we feel are necessary to build an effective automated IDEF4 tool.

4.1 IDEF4 Concept Requirements

The concepts that exist within IDEF4 will be familiar to those with object-oriented experience. The same structures that exist in most object-oriented languages also exist in IDEF4. The most notable concepts in IDEF4 are:

Classes,
Class Invariant Data Sheets
Features,
Inheritance Links,
Type Links,
Method Sets, and
Contract Data Sheets.

This section presents these basic structures and outlines the operations necessary to manipulate effectively these structures in an IDEF4 model.

Often these operations, especially the Copy operations, may be implicit in the tool as well as explicit. As will be discussed in Section 4.2, many different views of the same information may exist within the IDEF4 model. Because of this, it is reasonable to assume that these entities will be used repeatedly in different views. By referencing these entities in different views, the user is taking advantage of an implicit copy operation. The explicit operation should be made available as well, so that two entities that are similar in structure can be rapidly entered by creating one, copying that one, and then changing the copy to represent the second one.

4.1.1 Class Requirements

The class is the basic syntactic unit in an IDEF4 design model. The characteristics of a class are represented by a collection of features (see Section 4.1.2). Each feature can be either public or private, where a public feature is accessible to all classes and a private feature is accessible only by the class and its subclasses. The power of the object-oriented paradigm comes through the inheritance of classes. When an inheritance relationship is specified, all features of the parent class (superclass) are passed on to the child class (subclass). When this occurs, the inherited features in the subclass maintain the same characteristics as in the superclass unless they are explicitly redefined. This inheritance provides the ability to build complex class structures from simple classes.

Figure 8 shows how a class would appear in an IDEF4 Class Inheritance Diagram (see Section 4.2.1). The class is represented by a square-cornered box with the name of the class listed below the double line at the bottom of the box. IDEF4 requires that the first letter of the class name be capitalized. The features (see Section 4.1.2) of the class are also displayed in the class box with private features displayed below the export line and public features displayed above the export line. The feature symbols provide additional information about the role that the particular feature plays.

Listed below are the various functions necessary in an automated tool to manipulate effectively classes in an IDEF4 design model.

4.1.1.1 *Create Class*

The tool should support the creation of a class. At the time of creation of a class, the user must specify a unique name for the class to be created. Also at this time, the user may input feature and inheritance information. If this information is provided, the system must check the information to ensure that no conflicts arise. For example, if the user defines a class with a feature that it also inherits, the system must check that this feature has been tagged as redefined in this class and, if not, should automatically tag the feature as redefined.

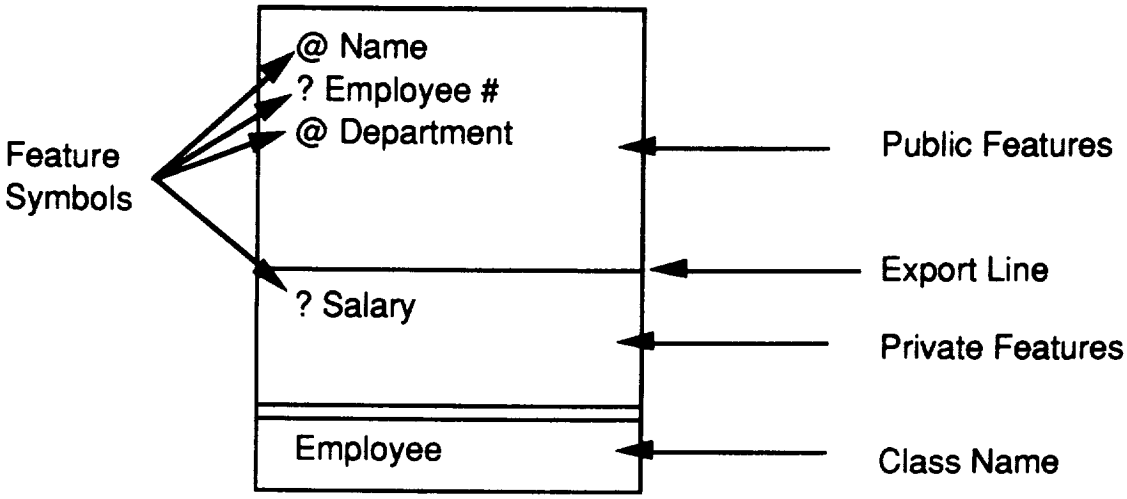


Figure 8. A Class Box

4.1.1.2 *Edit Class*

Any time after a class has been created, the Class can be edited. This might involve changing or editing the features associated with the Class, adding, editing or removing an inheritance link, or simply changing the name of the class. Once the user has completed the changes, the modifications will be examined to determine if any problems exist. If everything is valid, the changes will be reflected in the model, and all views appropriately updated.

4.1.1.3 Copy Class

The tool should also allow the structure of a class to be copied. When this operation occurs, a new Class is created by the system and the contents of the original class are copied directly into the new class. The only requirement is that the user give a different name to the new class. This operation will promote rapid development of the class hierarchy by allowing similar classes to be created from each other.

4.1.1.4 Delete Class

At any time after creation of a class, the user should be able to delete the Class. To perform this operation, the user must specify if the class is to be removed from the entire model or from the current diagram. If the class is to be removed from the current diagram, then any reference to the deleted class is removed from the diagram. Otherwise, every reference to the class in the entire model is removed.

For each class defined, the user can optionally attach a Class Invariant Data Sheet. This sheet is used to provide additional information about the objects in a class. The information represented in this data sheet must be true for all objects in the class at all times. An interesting feature of the Class Invariant Data Sheet is that subclasses also inherit the Data Sheet of their superior.

4.1.1.5 Create Class Invariant Data Sheet

The tool should support creation of a Class Invariant Data Sheet. After the user specifies the class to which the data sheet is to be attached, a reference to the data sheet is added to the class.

4.1.1.6 Edit Class Invariant Data Sheet

Any time after the creation of the Class Invariant Data Sheet, the user should be able to edit the data sheet. Once the appropriate changes are made to the text, the data sheet attached to the class is updated as well as the data sheets attached to classes that inherit from the modified class.

4.1.1.7 Copy Class Invariant Data Sheet

The user should also be able to copy Class Invariant Data Sheets. Because similarity is inevitable in a class hierarchy, it is very possible that two classes might have the same or similar invariant data sheet. For that reason, a copy of a data sheet should be available. All the user must do is specify the sheet to be copied and the class to attach the copy to.

4.1.1.8 Delete Class Invariant Data Sheet

Any time after creation of a Class Invariant Data Sheet, the user may delete that sheet. When this occurs, the reference in the Class to the data sheet is removed. In addition, appropriate action should be taken for any data sheets that inherit their content from the class whose data sheet has been deleted.

4.1.2 Feature Requirements

A feature is the named representation of a particular characteristic of a class. The features are used to capture the behavior of instances of a particular class. When the designer defines a feature, the type of the feature must be specified. It is important to distinguish between the type of the feature and the type of the value of the feature. Feature value type is concerned with the legal values that a feature may take or return. Feature type is concerned with the role that a feature will play within the context of a class. A feature can be only one of six types in an IDEF4 design model:

Feature (no symbol specified)
% Routine
? Attribute
$ Function
# Procedure
@ Slot

A Routine is any feature that initiates the execution of a block of code when queried. Also, a Routine can be classified as either a Procedure or Function. Procedure routines perform some side-effecting operation when executed and Functions return values. An Attribute is any feature that returns a value when queried. In addition, an Attribute can be classified as a value returning Function or a Slot, which is simply a variable location in machine memory.

Figure 9 shows the relationship between these feature types. The relationship represented by the double arrow is the Can Be/Is-a relationship, where a FEATURE can be an Attribute or a Routine and an Attribute is a FEATURE. FEATURE represents the most general specification whereas Procedure, Function, and Slot represent the most specific declarations. Note that there is no feature symbol for FEATURE. Any unlabeled feature defined in a class is automatically defined as a FEATURE.

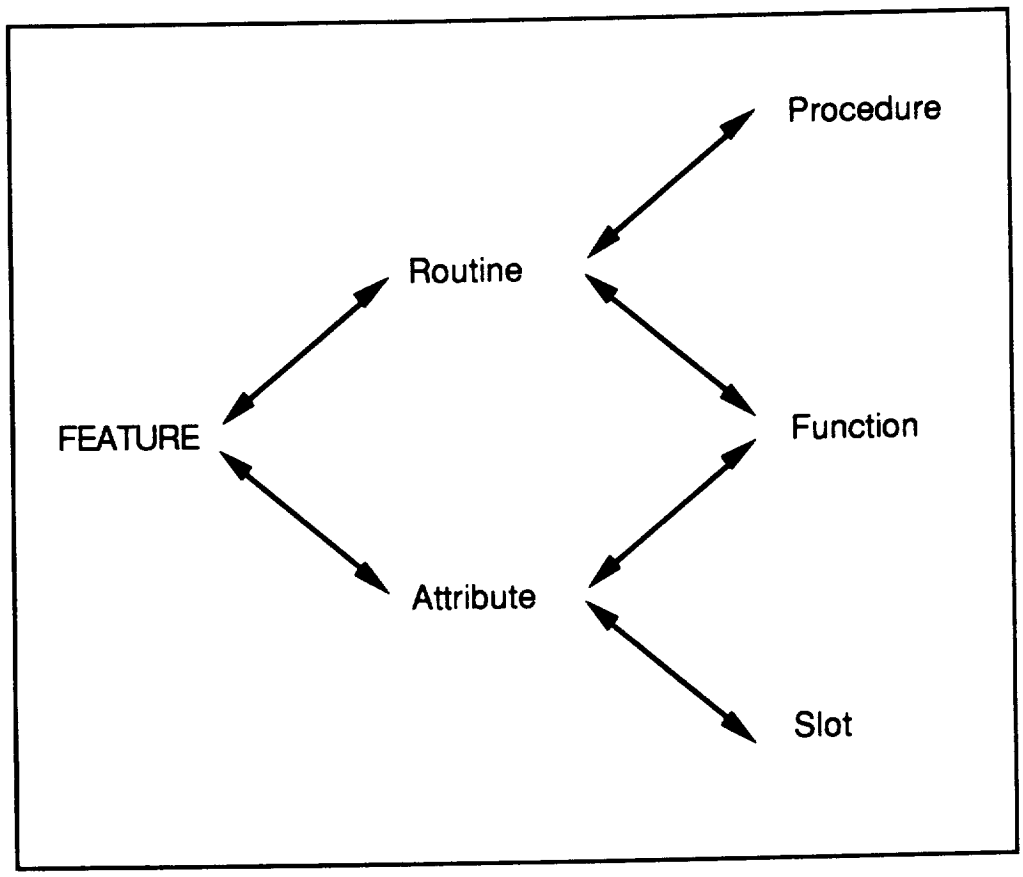


Figure 9. Feature Type Relationships

As was noted previously, features can be inherited through class inheritance links. However, since IDEF4 allows multiple inheritance (having multiple superclasses), it is possible for a class to inherit the same feature from two different classes. Also, it is possible for the characteristics of the feature to change. When these situations occur, the designer must redefine the feature to reflect these changes in character.

These changes are specified through the use of the auxiliary feature symbols shown below.

& Additional Contract
! New Taxonomic Specification
+ New Method
^ Now Public
* Now Private

These auxiliary symbols are placed to the left of the feature symbols in the box representing the class. The Additional Contract symbol (&) is used when a feature adds a contract to the method set to which the class-feature pair dispatches. When the same feature is inherited from two superclasses, the inherited feature must be redefined in the subclass inheriting the feature. The New Taxonomic Specification (!) is used to make this redefinition. The ! symbol also allows an inherited feature to be redefined more specifically (i.e., redefining an Attribute as a Function or Slot). The New Method (+) symbol reflects that the feature is to dispatch to a new method set instead of to the method set to which the feature in the superclass dispatches. If an inherited feature disagrees with its parent as to the features visibility, the Now Public or Now Private symbol must be used along with placing the feature on the correct side of the export line.

4.1.2.1 Create Feature

The tool should allow Features to be created. To perform this operation, the user must specify the name of the feature and the class where the feature will be attached. Also, at the time of creation, the user may optionally attach feature symbols or auxiliary feature symbols to the feature and specify the type, as well as the type display mode (textual or link) of the feature.

4.1.2.2 Edit Feature

At any time after a feature is created, the Feature should be accessible to editing. This operation might involve changing the name of the feature, changing the class to which the feature is attached, changing or adding feature symbols, or changing or specifying the type of the feature.

4.1.2.3 Copy Feature

Any time after a feature has been created, the user should be able to copy the Feature. This would be useful for reusing a feature in several different classes that do not inherit the feature. By creating the feature once, and setting up its type, the user can simplify the work of recreating the feature by simply producing a copy of the feature and indicating the new class where the copy will belong.

4.1.2.4 Delete Feature

Any time after a Feature has been created, the Feature should be able to be deleted. If this operation is performed, any reference to this feature in a class should be removed. In addition, if any classes inherit this feature from this class, the subclasses should be checked to see if any redefinitions of the feature occur and any corrective procedures executed.

4.1.3 Inheritance Link Requirements

An Inheritance Link simply defines a parent/child relationship between two classes. When an inheritance link is specified, all characteristics of the parent class are inherited in the child class. Also, the inherited features will exhibit the same behavior in the child class as in the parent class unless the features are redefined using the auxiliary feature symbols.

4.1.3.1 Create Inheritance Link

The tool should support the creation of Inheritance Links. The link will specify a subclass/superclass relationship between two classes. Inheritance Links can be created both implicitly and explicitly. If during creation of a class, that class's superclass is specified, an implicit inheritance link has been created. However, if inheritance is specified between two existing classes, then an explicit creation has occurred.

4.1.3.2 Edit Inheritance Link

Any time after an Inheritance Link has been created, it should be possible to edit that link. This operation would simply change the two classes related by the inheritance

link, and then update the model to reflect the change in the class hierarchy.

4.1.3.3 Copy Inheritance Link

The tool should also allow Inheritance Links to be copied. However, when a link is copied, the system must require that at least one of the classes related by the link be changed. Otherwise, a duplicate link will exist within the system.

4.1.3.4 Delete Inheritance Link

Any time after an Inheritance Link has been created, that Link can be deleted. This deletion could have a tremendous effect on the class hierarchy so it is important that the system correctly update the model to reflect the changes made by deleting this link.

4.1.4 Type Link Requirements

A class can be considered to be a data type, and traditional programming data types can be considered to be classes. Since the features of classes that are classified as Attributes, or more specifically Slots or Routines, take on values, it would be useful to indicate the type of the value that the feature will take. These type declarations are made with Type Links. The Type Link specifies the feature being typed and the class that represents the legal values that the Attribute may take.

Figure 10 shows the four different Type Links supported in IDEF4. The first link simply says that the Attribute f of A returns a value of Type B. The second link is identical to the first except there is also a dual: while f of A returns a value of Type B, g of B returns a value of Type A. This dual link reduces the number of links that might appear in a Type Diagram and thus provides for a simpler diagram. The third link indicates that f of A returns a value that is constructed from B. This may be a list of instances of B or some other structure composed of instances of B. Finally, the fourth link provides a semi-dual for the third link type. While f of A returns a value constructed from instances of B, g of B returns a value of Type A.

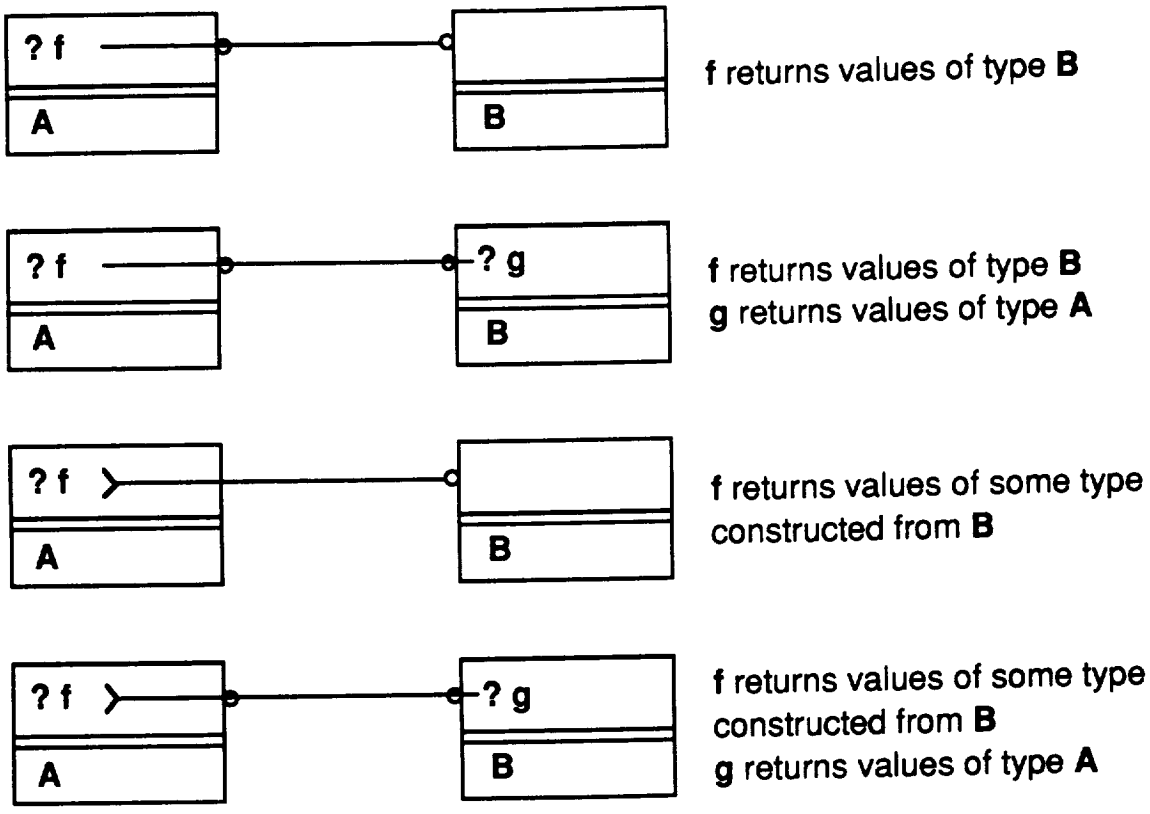


Figure 10. IDEF4 Link Types

In addition, a Type Link has two possible graphical presentations in the Type Diagram (see Section 4.2.2). One presentation is graphical such as in Figure 10 where a line physically links the feature with the class representing the type of that feature's value. The second is textual, where the feature name is followed by a colon followed by the name of the type that the feature's value can take. The user decides which presentation is most appropriate in presenting the type information. It is also possible for both display methods to be used.

4.1.4.1 Create Type Link

The tool should support the creation of Type Links. This could be either (1) an implicit operation by specifying the feature type when the feature is created or (2) an explicit operation where the user specifies the feature, the type to be linked, and the presentation method to be used for displaying the link.

4.1.4.2 Edit Type Link

Once a Type Link is created, it should be possible to edit that link. This may involve changing the type or changing the display method of the link. Any changes to the Link should be reflected in any of the Type Diagrams (see Section 4.2.2) that display this particular type link.

4.1.4.3 Copy Type Link

The tool should also allow type links to be copied. The only requirement in the copy operation is that the feature that is typed by the link must be changed to a new feature. Otherwise, a duplicate type declaration for the feature will exist in the system.

4.1.4.4 Delete Type Link

After a Type Link has been created, the Link can be deleted. By deleting this link, the feature's type will return to being labelled as unspecified. Any Type Diagram using the deleted type link will be updated to reflect this deletion.

4.1.5 Method Set Requirements

IDEF4 does not represent individual methods. The reason for this is that a method could accept parameters that are instances of more than one class. As a result, the same method must be defined for both classes. To alleviate this repetition and confusion, the information of these methods will be maintained in a method set. A feature of a class and that class will map to a method set. This mapping is referred to as Dispatch Mapping.

Figure 11 shows how a method set would appear in a Method Taxonomy Diagram. The box represents the set, with the method set name appearing in the center. The information in the square brackets is a list of class/routine feature pairs that map to this particular method set. This bracketed information is optional.

A method set is defined by its associated contract. In actuality, a method set is just a name for a contract data sheet. The contract data sheet maintains the declarative statements that define the intended effect of the methods in the method set. For a function, the contract would state the relationship between the function's argument list and the corresponding return values. For a procedure,

the contract would have to specify how the method set changed the environment when passed an argument list and the current environment.

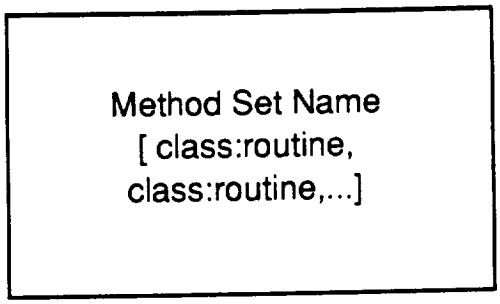


Figure 11. Method Set Box

4.1.5.1 Create Method Set

The tool should support the creation of Method Sets. This operation would require that a set name be given. Optionally, the user may specify for dispatching purposes the class/routine pairs that map to this set.

4.1.5.2 Edit Method Set

Once a method set has been created, the user should be able to edit the method set. This might involve changing the name of the set or changing the class/routine pairs that map to this set. When this operation is completed, the changes should be reflected in the method set and all diagrams that reference this method set should be updated.

4.1.5.3 Copy Method Set

The tool should allow a method set to be copied to reduce the amount of repetitive data entry. The only requirement when copying a method set is that the name of the copied set must be changed before the copy operation can be completed.

4.1.5.4 Delete Method Set

Once a method set has been created, that method set can be deleted by the designer. When this operation is

performed, the method set is removed from the method submodel and all diagrams that reference the deleted set are updated to reflect its absence.

Operations are also required for the contract data sheet. Since a method set simply provides a front-end to the contract data sheet, these operation may be embedded in the operations for the method set outlined above.

4.1.5.5 Create Contract Data Sheet

The tool should allow the user to create a Contract Data Sheet. This operation would require the user to specify the method set where the data sheet is to be attached.

4.1.5.6 Edit Contract Data Sheet

Once a Contract Data Sheet has been created, that data sheet can be edited by the user. This would involve adding, deleting, or changing contracts maintained in the data sheet.

4.1.5.7 Copy Contract Data Sheet

Once a Contract Data Sheet has been created, that data sheet can be copied and attached to a different method set. This operation requires the user to specify the data sheet to be copied and the method set where the copy is to be attached.

4.1.5.8 Delete Contract Data Sheet

Once a Contract Data Sheet has been created, that data sheet can be deleted. When this operation is performed, the reference from the method set to this data sheet is removed.

4.2 IDEF4 Diagram Requirements

Conceptually, all information in an IDEF4 model is represented in the Class Submodel and Method Submodel, with the Dispatch Mapping linking the two together. However, due to the large size of these submodels, it is necessary to view the information in various diagrams in order to understand effectively the information stored in an IDEF4 model, . These diagrams include:

Inheritance Diagrams
Type Diagrams
Protocol Diagrams
Method Taxonomy Diagrams
Client Diagrams

Though the type of information in each of the diagrams is restricted, the actual content of each diagram is left for the user to decide. In other words, the user can decide exactly what objects are displayed in each diagram, except for the Protocol Diagram whose simplicity does not allow this flexibility. Also, since the user can configure these diagrams, it becomes necessary to provide the ability to maintain several of each type of diagram. In essence, the system will be maintaining several different views of the same information. This section will describe the operations necessary to create and maintain these different views of the information stored in the IDEF4 model.

4.2.1 Class Inheritance Diagram Requirements

A Class Inheritance Diagram presents the inheritance relationships that have been defined between classes in the model. Figure 12 shows an example Class Inheritance Diagram. This diagram presents the inheritance relationships that have been defined between classes in the model. The boxes in the diagram represent the classes with the arrows representing the inheritance relationship, pointing from the parent to the child class. As the number of classes in a model can be quite large, it would be impractical to include all the classes in one inheritance diagram. Instead, several different Class Inheritance Diagrams can be defined by the designer where each diagram centers on a different group of closely related classes.

4.2.1.1 Create Class Inheritance Diagram

The tool should allow the user to create a Class Inheritance Diagram. In performing this operation, the user is creating a special view of the entire class submodel. As a result, the user must specify the classes to be examined in this view. The classes specified by the user may not be the only classes displayed in the diagram as it may be necessary to include other classes in the diagram to display adequately the inheritance relationship. The tool should have the ability to determine when these additional classes are necessary. Also, since there may be many views, a

unique name must be given to the diagram so that the system can distinguish between the different views.

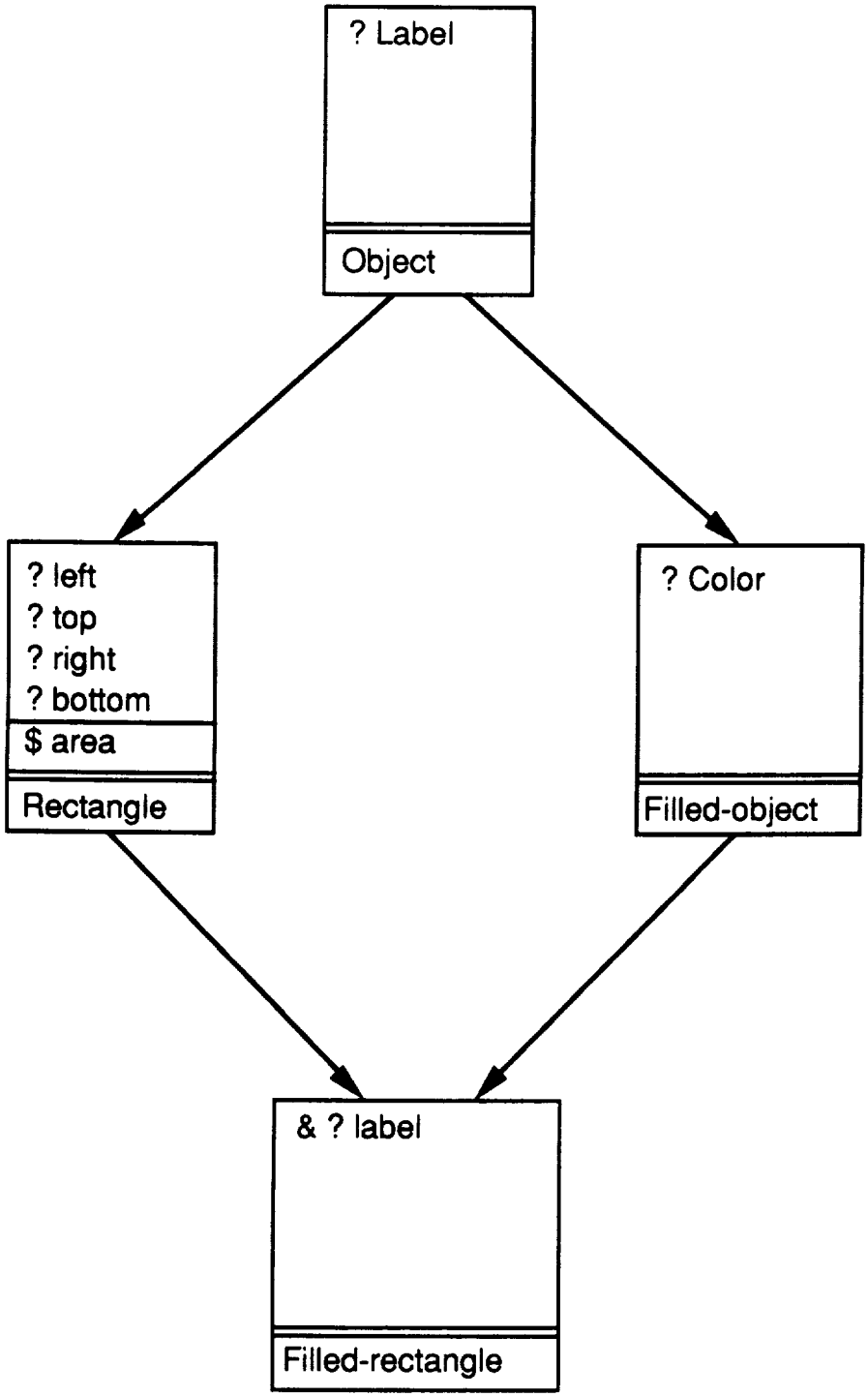


Figure 12. IDEF4 Inheritance Diagram

4.2.1.2 *Edit Class Inheritance Diagram*

Any time after an Inheritance Diagram has been created, the user should have the ability to edit the diagram. This would mainly involve changing the classes that are examined in the diagram or changing the inheritance structure of the classes in the diagram. When the changes have been completed, the diagram should be immediately updated. Also, any changes in the inheritance structure should be reflected in all the diagrams that the change would effect. The tool should also ensure that the change does not have any adverse effect, such as unexpected redefinition of features of classes through inheritance. If something like this is found, the user should be notified and should be required to take action to rectify the situation.

4.2.1.3 *Copy Class Inheritance Diagram*

The tool should also provide the ability to copy an Inheritance Diagram into another diagram or copy the diagram as the base for a new Inheritance Diagram. This would allow for the rapid development of similar diagrams, since one diagram could be built from another diagram. If the user is copying a diagram to initiate a new diagram, the user must specify a new name for the new diagram.

4.2.1.4 *Delete Class Inheritance Diagram*

Any time after an Inheritance Diagram has been created, the user should have the ability to delete the diagram. When this operation is performed, the diagram should be removed from the screen and any reference to the diagram should be removed from the model.

4.2.2 Type Diagram Requirements

A Type Diagram captures the type information associated with features of a class. This type information is restricted to features that are classified as attributes (?), or more specifically slots (@) or functions ($). Also, auxiliary feature symbols and the export line are not used in this diagram.

Figure 13 shows a sample IDEF4 Type Diagram. Notice that this diagram uses both the textual and graphical type link display. The

display mode of the type link is defined by the modeler, but both are used here for demonstration purposes.

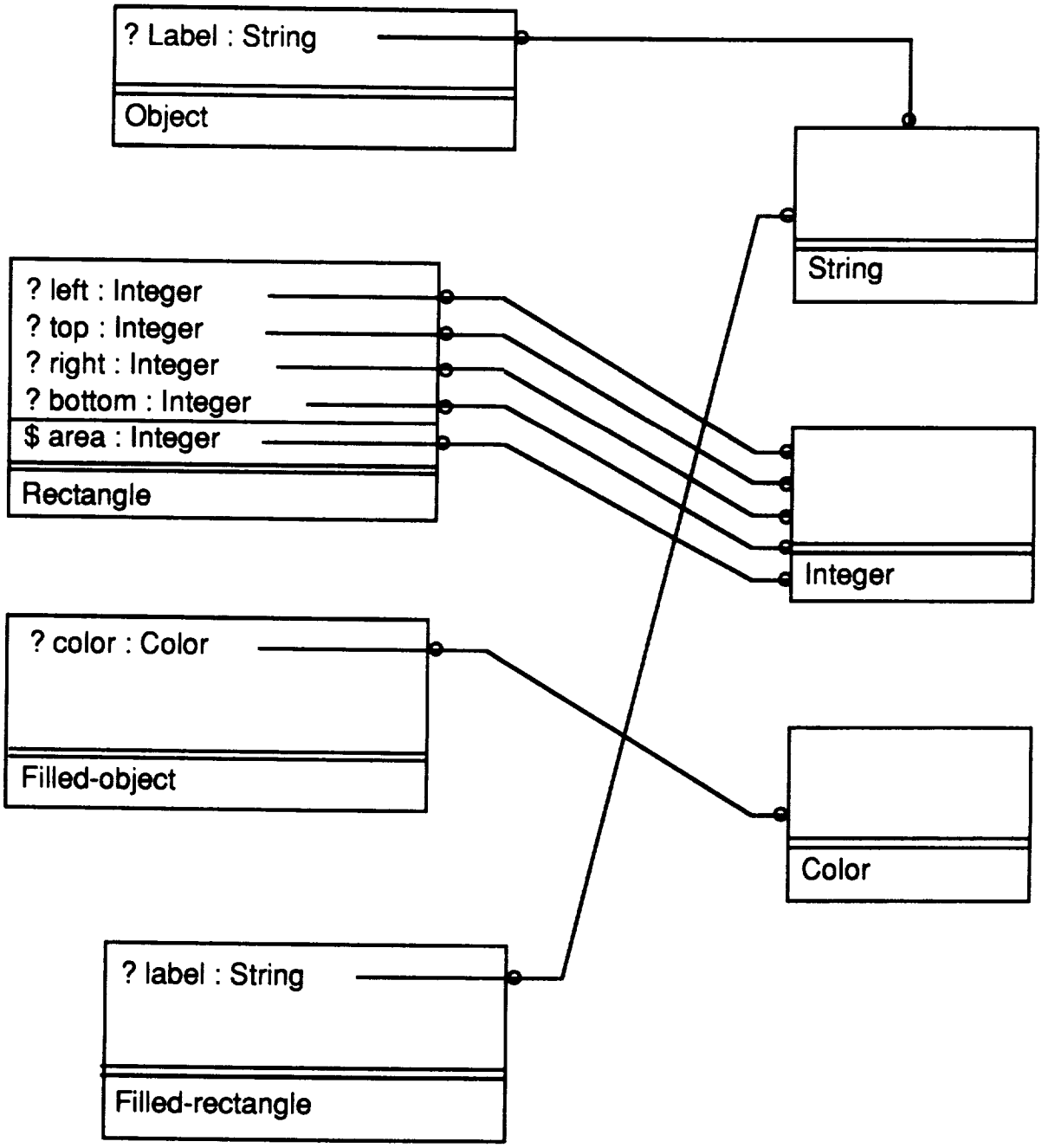


Figure 13 IDEF4 Type Diagram

4.2.2.1 Create Type Diagram

The tool should allow the user to create a Type Diagram. Since several different Type Diagrams may exist as separate views of the same model, it is necessary for each diagram to be given a unique name. In addition, the user must specify the classes whose features are to be the focus of this new type diagram. The classes specified by the user may not be the only classes displayed in the diagram, since it may be necessary to include other classes in the diagram to display adequately the type relationships. The tool should have the ability to determine when these additional classes are necessary.

4.2.2.2 Edit Type Diagram

Once a type diagram has been created, the user should have the ability to edit the diagram. On a high level, this would involve changing the classes that are displayed in the diagram. But, on a lower level, the user should have access to the features associated with a class displayed in the diagram. This would allow editing of that feature, possibly to change the type of the feature or to change the display characteristic to the type link associated with the feature. Any changes to classes or features in a type diagram should be automatically reflected in all other diagrams with the class or feature is associated.

4.2.2.3 Copy Type Diagram

Once a type diagram has been created, the user should be able to copy the diagram. This operation would copy the contents of the specified diagram into another diagram and would allow for rapid development of different views of the model by reducing the amount of redundant work that must be performed.

4.2.2.4 Delete Type Diagram

Once a Type Diagram has been created, the diagram can be deleted from the design model. When this occurs, the diagram should be removed from the screen and any reference to this diagram should be removed from the design model.

4.2.3 Protocol Diagram Requirements

A Protocol Diagram maintains information on the argument list of an Attribute, Slot, or Function feature of a class. The diagram lists the arguments that the designer has specified for the feature. In addition, the type of the argument is displayed in a fashion similar to that of the Type Diagram. The type links in this diagram have the same meaning as the links in the Type Diagram.

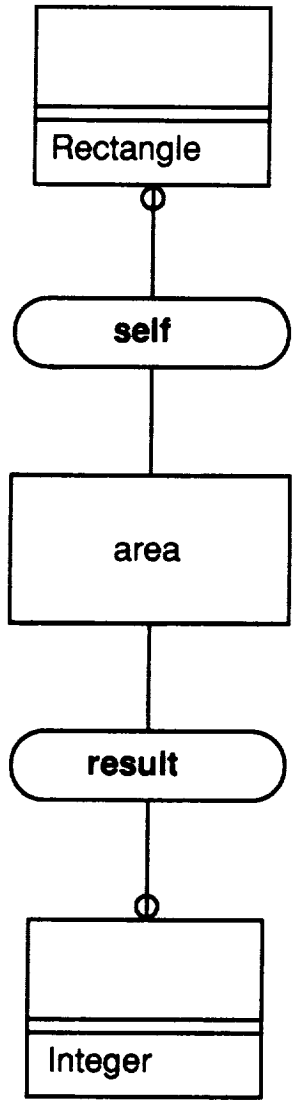


Figure 14. IDEF4 Protocol Diagram

Figure 14 shows a sample Protocol Diagram for the Area routine of the class Rectangle. The center box represents the feature whose protocol is being specified; the boxes with rounded corners represent arguments; and the class boxes represent the types of the arguments. Every protocol has two default arguments, 'self' and 'result'. These objects represent an instance of the class that the feature is a part of and the value returned by the feature respectively.

Technically, by default, every feature of a class will have a Protocol Diagram that will simply consist of the feature name with an argument of 'self' and a return type of the type specified in the Type Diagram. However, it should be left to the implementation to determine (1) if this default protocol is to be created automatically at the time a feature is created or (2) if the protocol is to be created by an explicit operation performed by the modeler.

4.2.3.1 Create Protocol Diagram

The tool should support the creation of a Protocol Diagram for a feature of a class. When this occurs, the protocol will be given an the default Self argument and a return type of the type specified in the Type Diagram for the feature and class.

4.2.3.2 Edit Protocol Diagram

The user should have the ability to edit a protocol diagram. This would involve adding, removing, or editing the arguments to the feature protocol or changing the type of an argument in the feature protocol.

4.2.3.3 Copy Protocol Diagram

The user should have the ability to copy a protocol diagram. When this operation is performed, the specified protocol diagram will be copied and then attached to a feature associated with a class. The tool should automatically update the new copy to reflect the feature to which the copy has been attached (i.e., change the name).

4.2.3.4 Delete Protocol Diagram

The user should have the ability to delete a protocol diagram associated with a feature of a class. As with the *Create Protocol Diagram*, this operation will depend on the implementation. If every feature has a default protocol diagram upon creation of the feature, then the deletion

would simply replace the feature's current protocol diagram with this default diagram. If a protocol diagram is associated with a feature only through an explicit operation, however, the deletion would remove any reference to a protocol diagram from the feature.

When building a protocol diagram, it is necessary to add arguments to the protocol of the feature. The following operations provide the capability required to manipulate effectively these arguments.

4.2.3.5 Create Argument

The Create Argument operation should allow arguments to be specified for the protocol of a feature. During this operation, the name and the type of the argument should be specified.

4.2.3.6 Edit Argument

The Edit Argument operation should allow the arguments of a protocol diagram to be updated. This would mainly involve changing the name or the type of the argument being modified.

4.2.3.7 Copy Argument

The Copy Argument operation should allow the user to make copies of an argument and attach the copy to the protocol diagram. The copy would keep the same argument type as the original, but would require the user to change the name of the argument.

4.2.3.8 Move Argument

The Move Argument operation should allow the user to reorder the arguments of a particular protocol diagram.

4.2.3.9 Delete Argument

The Delete Argument operation should allow the user to remove an argument from the protocol diagram for a feature. However, the system should not allow the Self argument to be removed at any time.

4.2.4 Method Taxonomy Diagram Requirements

The Method Taxonomy Diagram is used to show relationships among method sets that are independent of the class hierarchy. The

Diagram consists of method sets, represented by boxes as in Figure 11, and links between the method sets. The links in this diagram represent the pure superset/subset relationship, and the arrow points from the superset to the subset. The power of these diagrams is seen when a particular contract has been widely used and studied. For example, the different sorting methods can form a very complex taxonomy, and that taxonomy can serve as a valuable resource to the designer.

4.2.4.1 Create Method Taxonomy Diagram

The tool should support the creation of Method Taxonomy Diagrams. When this operation is performed, the designer must specify the method sets to be included in the taxonomy.

4.2.4.2 Edit Method Taxonomy Diagram

Once a Method Set Taxonomy Diagram has been created, the user should be able to edit that diagram. This operation would involve adding or deleting method sets to the taxonomy.

4.2.4.3 Copy Method Taxonomy Diagram

The designer should be able to copy Method Taxonomy Diagrams. This would promote rapid development of the diagrams by allowing new diagrams to be created by copying and then making minor modifications to the copy to arrive at a completed diagram.

4.2.4.4 Delete Method Taxonomy Diagram

Once a Method Taxonomy Diagram has been created, the user should have the ability to delete that diagram. When this occurs, the diagram will be removed from the Method Submodel.

To complete Method Taxonomy Diagrams, the following operations for the superset/subset relationship must be provided.

4.2.4.5 Create Method Set Link

To represent a superset/subset relationship between two method sets in a Method Taxonomy Diagram, the tool must support the creation of Method Set Links. This operation would require that the designer specify the two method sets to be related by the link.

4.2.4.6 *Edit Method Set Link*

Once a Method Set Link has been created, the designer should be able to edit that link. This operation would involve the user changing the method sets that are related by the link.

4.2.4.7 *Copy Method Set Link*

The tool should allow a Method Set Link to be copied by the designer. This operation would require that the designer change at least one of the method sets related by the original link before the copy operation can be completed.

4.2.4.8 *Delete Method Set Link*

Once a Method Set Link has been created, the designer should have the ability to delete that link. When this occurs, any relationship between two method sets represented by the link will be removed from the Method Submodel.

4.2.5 Client Diagram Requirements

The Client Diagram is IDEF4's version of the structure chart. It presents the caller/callee (client) relationship between routines and usually centers on one particular (client) routine. In the diagram, a routine is represented by a rectangle and a call is represented by a line with double barbs at both ends that point from the called routine to the calling routine. The routine box also the routine/class pair displayed in the center, with the two name separated by a colon. In cases where the called routine is defined in several classes, the Client Diagram only displays one of the routine/pairs.

4.2.5.1 *Create Client Diagram*

The tool should support the creation of Client Diagrams. When this operation is performed, the designer should specify the routine/class pair that will serve as the client and focus of the diagram. The user must also specify the routines that are called by the client and any routines that call the client.

4.2.5.2 *Edit Client Diagram*

Once a Client Diagram has been created, the designer should have the ability to edit the diagram. This might

involve changing the actual client, the routines that are called by the client, or routines that call the client.

4.2.5.3 Copy Client Diagram

The designer should have the ability to copy Client Diagrams. The only requirement of this operation is that the user must change the Client routine before the copy operation can be completed.

4.2.5.4 Delete Client Diagram

Once a Client Diagram has been created, the designer should be able to delete that diagram. When this occurs, the diagram will be removed from the Method Submodel.

4.2.6 Customized Diagram Requirements

The previous sections have described the "pure" diagrams that must be supported by an IDEF4 tool implementation. However, it would also be advantageous to allow the user to build customized views of the design model by combining information maintained in several diagrams in a single diagram. An example might be to include the dispatching information present in the Method Taxonomy Diagram in the Class Hierarchy Diagram. Figure 15 shows how a the Rectangle class would appear in the diagram if the dispatching information for the area routine were included.

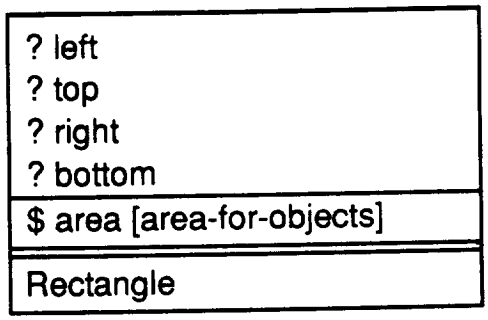


Figure 15. Class with Dispatching Information

To facilitate these mixed mode diagrams, the tool should provide the ability to generate customized diagrams. The following are two diagrams that we have identified as being commonly used in an IDEF4 design model. It is possible that more diagrams of this type

would be useful. As a result, the tool should make every effort possible to provide for the building of these customized diagram.

4.2.6.1 Class Inheritance/Dispatching Diagram

This diagram combines the class inheritance diagram with the method set taxonomy diagram. The diagram will look very similar to the class inheritance diagram except that routine features that have been dispatched to method sets will be followed in the class box by the method set name in square brackets. Since this diagram is very much like the class inheritance diagram, the operations for creation and manipulation are nearly identical to the operations for the class inheritance diagram (see Section 4.2.1).

4.2.6.2 Class Inheritance/Type Diagram

This diagram combines the class inheritance diagram with the type diagram. A diagram of this type will be identical to a class inheritance diagram, except that every feature will be followed by a colon and then its type as specified in the type diagram. In discussing the type diagram (Section 4.2.2), we mentioned the various display modes of the type links. This diagram will ignore the user specified display mode and present all type information in the textual format following the feature name. Since this diagram is very much like the class inheritance diagram, the operations for creation and manipulation are nearly identical to the operations for the class inheritance diagram (see Section 4.2.1).

4.3 IDEF4 User Interface Requirements

Perhaps the most critical portion of a modeling tool is the User Interface. The goal of any automated tool is to make the process being automated easier to perform with the tool than without the tool. To meet this goal in the IDEF4 tool, it is important that certain capabilities be available to the user to promote easy movement and access to the various components of the diagrams.

The User Interface is especially important in an automated IDEF4 tool. The reason for this is that the IDEF4 design methodology centers around various views of a design model. To present the class inheritance structure would be difficult because of the potential size

of the diagram. Also, presenting the entire diagram would be useless since many classes are completely unrelated to other classes in the diagram. The views within a design model allow the user to present logical units of the design model. As a result, the ability to move easily between these diagram is very important.

4.3.1 Browsing Requirements

The most important aspect of the IDEF4 tool user interface is the display of the various diagrams being developed by the modeler and the information maintained in those diagrams.

4.3.1.1 Select Diagram

The IDEF4 design model will consist of several different "view" diagrams. As the number of these diagrams could grow quite large for large designs, it is important that the tool provide the ability to move between different diagrams easily. The Select Diagram operation will simply allow the user to display and examine a selected diagram.

4.3.1.2 Find Diagrams

In an IDEF4 design model, it is possible for an object (class, feature, etc...) to be used in several different diagrams. It would be useful if the tool provided the ability to query for all the diagrams in which a particular object appears. This would allow a thorough examination of all the uses of a certain object.

4.3.1.3 Speed Search

Due to the size of IDEF4 class diagrams, the tool should be equipped with a Speed Search utility. This function would allow the user to input the name of a particular object that they wish to examine, and the tool would redisplay the diagram or listing to display the object whose name most closely matches the partial string entered by the user. This is equivalent to string searching in a text editor, except that the diagram is being searched instead of a block of text.

The following capabilities represent alternative viewing modes of information maintained in an IDEF4 design model. While the various diagrams discussed in Section 4.2 provide an excellent mechanism for viewing an IDEF4 design model, these alternative modes provide

a more efficient method for initially inputting class information into the model. Instead of viewing the IDEF4 diagrams graphically, these modes may display information as a list of textual information. Each of these modes should support the operations described previously to allow for rapid browsing and editing of the information presented.

4.3.1.4 Class Listing

The Class Listing mode simply displays a list of the classes defined in the current model. Useful organization of the classes may be appropriate as well. Two possibilities might be simply an alphabetical listing of the classes or a more complicated indented hierarchy listing of the classes, where subclasses are indented below their superclass. It might also be useful to display the features associated with the classes.

4.3.1.5 Protocol Listing

The Protocol Listing will display a list of all the features defined in the design model along with the protocols defined for those features. An effective display should be developed so that there will be no confusion with which class a particular feature is associated.

4.3.1.6 Method Set Listing

The Method Set Listing will simply be an alphabetical listing of all the method sets that have been defined in the design model. Other useful information in this listing could be the classes that have defined contracts for the particular method sets.

4.3.2 Group Operation Requirements

To reduce the number of repeated operations, and thus to speed up the development of IDEF4 models, it would be useful if certain operations could be performed on groups of objects in the diagrams.

4.3.2.1 Copy Structure by Group

The tool should allow the user to copy the structure of several objects at one time. This operation would promote the rapid development of the design model. For example, if two groups of classes were related in an identical fashion, the class inheritance structure could be established for the first group of classes and that structure

could be copied within this operation. The tool would create new classes and maintain only the inheritance links between the new classes. The user would then add the names of the second group of classes to the copied structure.

4.3.2.2 Copy Information by Group

The tool should allow the user to copy several objects at one time. This operation would promote the rapid development of several views of the design model by allowing information displayed in one view to be easily copied into another view. When this operation is performed, no new objects are created. The selected objects are will simply be referenced in the new view. As a result, this operation will be used mainly in building up the different views of a design model that is fully populated with the design information.

4.3.2.3 Delete by Group

The tool should allow the user to delete several objects from a diagram as a single unit. After executing this operation, the selected objects will be removed from the current diagram. In addition, the diagram will be checked to see if any loose ends remain from the group deletion. If so, these conditions should be appropriately handled by the tool.

4.3.3 Report Generation Requirements

To assist in the development of reports surrounding an IDEF4 model, it is important that the following functions be supported.

4.3.3.1 Hardcopy

An automated IDEF4 tool should provide hardcopy support for the design models. This support should include printing of the various view diagrams as well as a listing of the classes and method sets that are a part of a design model. These hardcopies could then be included in system design documents or distributed to code developers to clarify design issues.

4.3.3.2 ASCII File Dump

The tool should provide the ability to generate an ASCII file of the information maintained in and IDEF4 design model. This would enable database applications to be generated for the information maintained in the IDEF4 designs.

4.3.3.3 Note Attachment

The tool should support the attachment of notes to design elements so that information about the structure can be captured. These notes would be organized in a HyperText fashion to allow the greates degree of flexibility.

4.4 IDEF4 Integration Requirements

At the writing of this requirements document, the integration requirements of the IDEF4 design methodology have not been completely defined. Nevertheless, the tool should provide a general integration facility. The most general strategy is a data extraction facility that provides the ability to extract information maintained in an IDEF4 design model for use in other applications. The designer should have the ability to indicate what information should be extracted as well as the structure that the extracted information should be produced in.

4.5 IDEF4 Information Management Requirements

Related to the development of models is the problem of model validation. Any automated tool should provide the means to ensure that models being created actually conform to rules and guidelines of the methodology. As a result, the tool should have the following capability.

4.5.1 Model Verification

The tool should have a working understanding of the IDEF4 Metamodel, a model of IDEF4 in IDEF1. This knowledge is imperative if the tool is to be effective. Without this information, the user would have free reign to develop any type of construct possible with the entities of IDEF4. By encoding the metamodel into the IDEF4 tool, the

development of valid IDEF4 designs is ensured by allowing only valid IDEF4 constructs to be created.

One of the most basic requirements, as well as the most important, is the ability to save and reload the information stored in a model. As a result, it is imperative that the IDEF4 tool provide these capabilities.

4.5.2 Model Saving

The tool should allow a model to be saved. At the completion of an modeling session, the user should be able to save the entire model. This would include saving all the view diagrams as well as the class submodel and method submodel.

4.5.3 Model Loading

The tool should allow a model to be loaded. If models have been saved, the user should be allowed to load any of those models into to the tool for browsing or editing purposes. More sophisticated tools could support access control so that only authorized users would be capable of loading a model in an edit mode. All other users would only be able to browse the model.

4.5.4 Model Database

The tool should support database-like operations to provide for the rapid extraction of information from the design model. The operations should allow the user to query against the design for information related to certain objects represented in the design.

The following two capabilities allow for a submodel or diagram to be used in other models. These utilities should allow a portion of one model to be written to a file and then later reloaded, as a submodel, into another model. This capability will greatly reduce the amount of work that must be duplicated.

4.5.5 Kit Save

The tool should allow a portion of model (kit) to be saved to a file. The model portion could be a specific diagrams or several classes or method sets.

4.5.6 Kit Load

The tool should allow a kit to be loaded into the current model. The user should specify whether the information stored into the kit is to be loaded as part of the class or method submodel or as a view diagram.

The final information management requirement deals with the life cycle of the design maintained in an IDEF4 design model. Often, a design loses its effectiveness over time because the programmers do not understand a particular aspect of the design and the designer is not around to explain why it was designed the way it was.

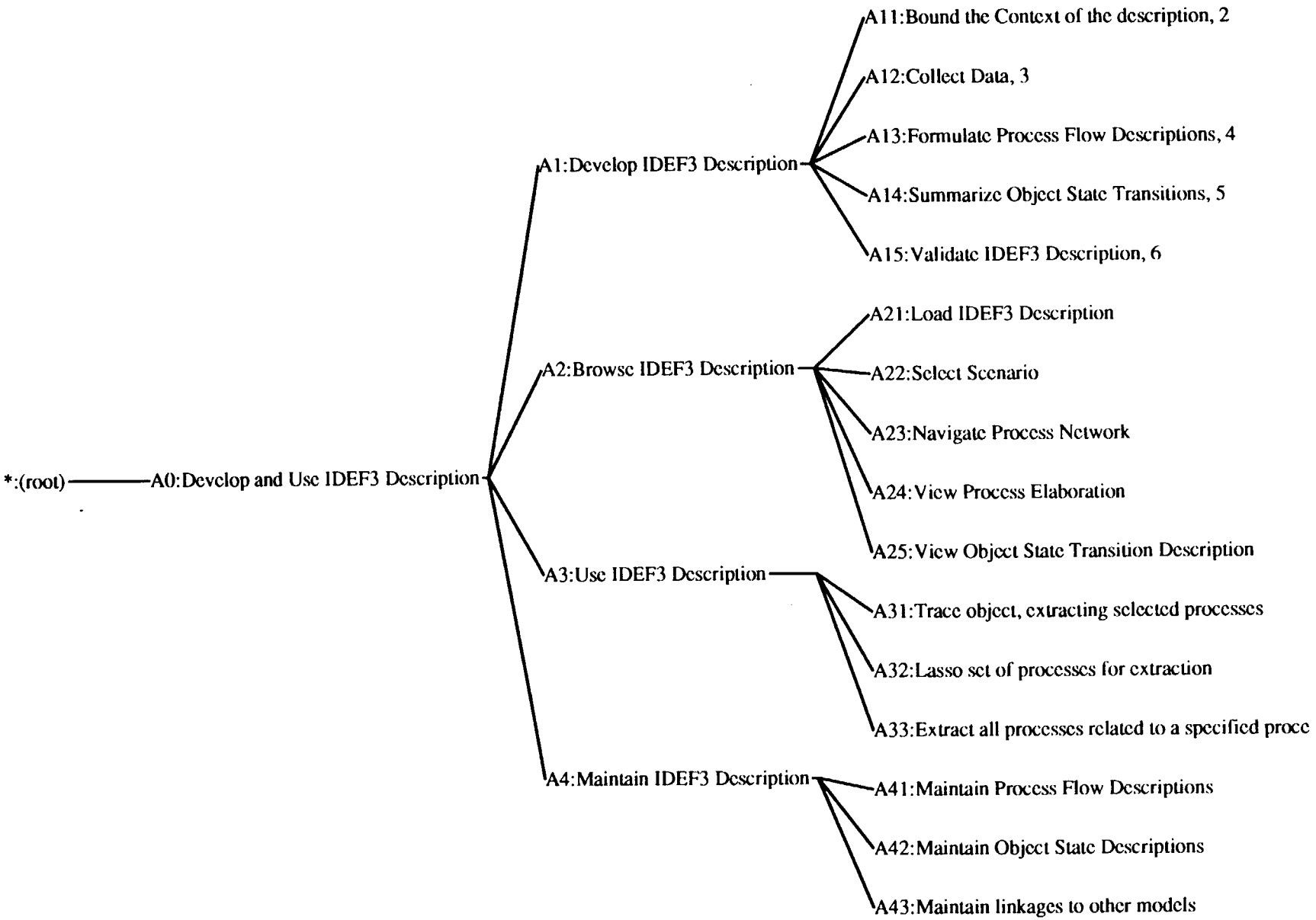
4.5.7 Design Life Cycle Management

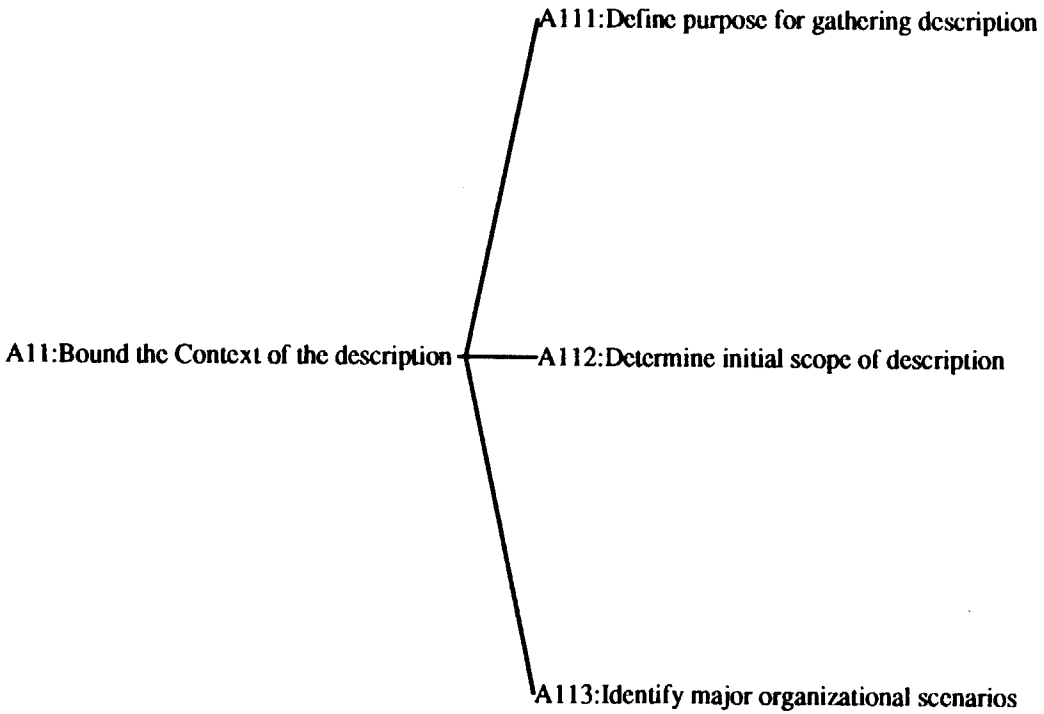
The tool should provide the means to effectively capture the designers intent, rationale, and philosophy of system operation. This information would provide a person studying the design with a greater understanding of how particular aspects of the design were arrived at.

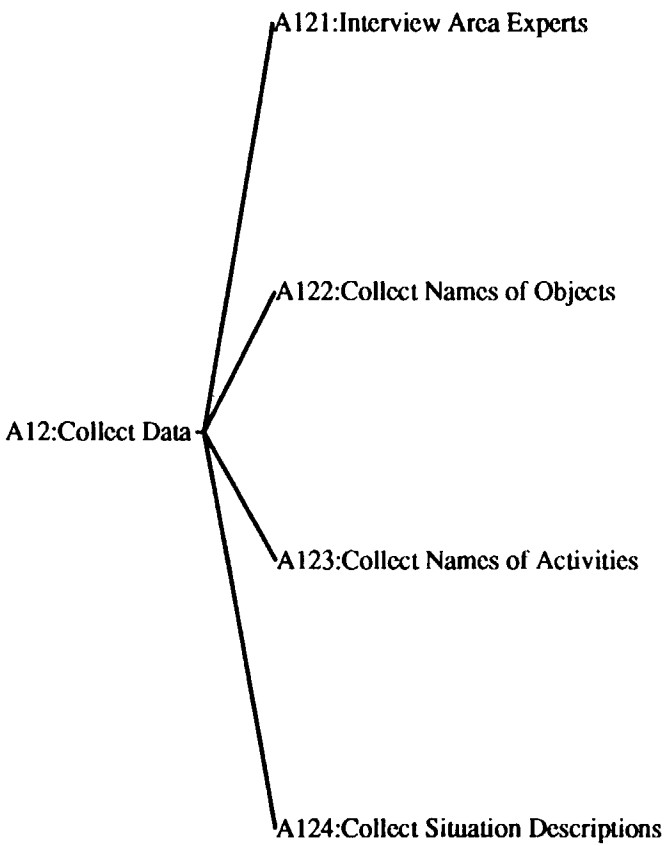
4.6. Summary

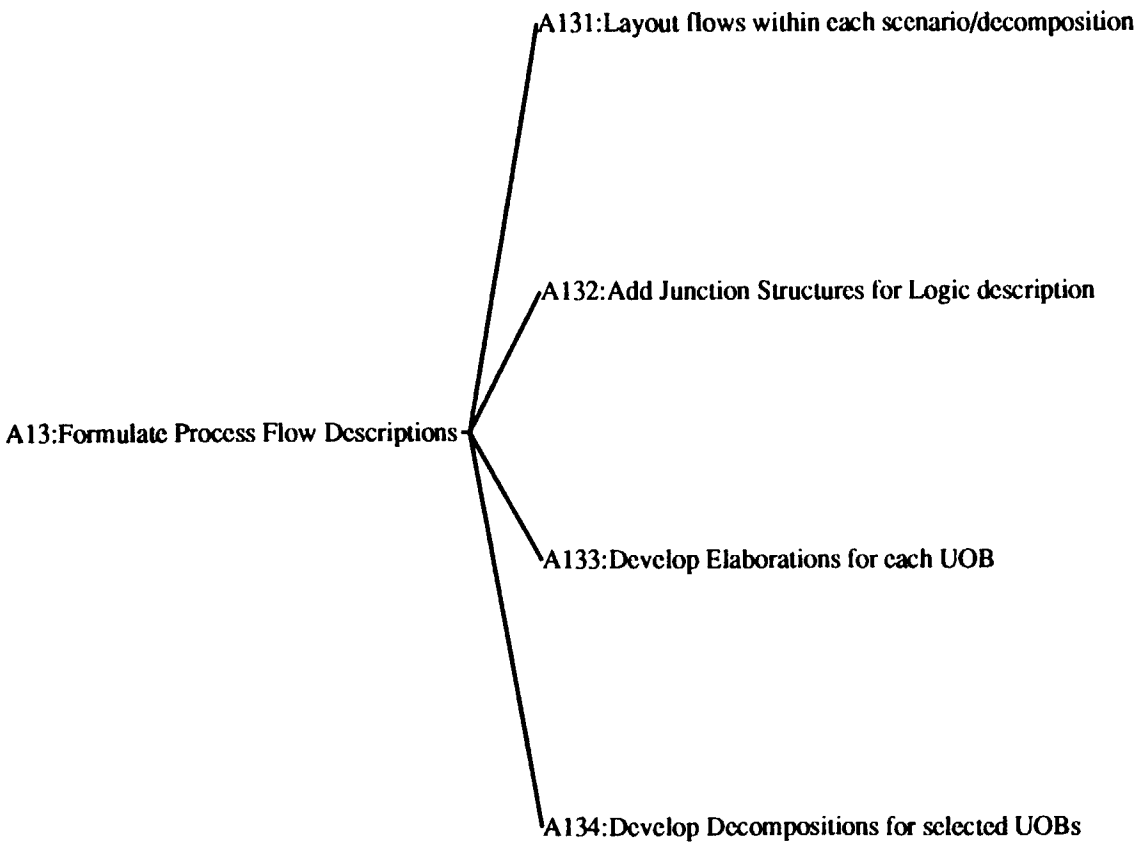
This section has presented the minimal functionality necessary for the development of an effective tool for automating the IDEF4 design methodology. Four major areas of functionality have been addressed: IDEF4 Concept Requirements, IDEF4 Organizational Requirements, IDEF4 Integration Requirements, and IDEF4 Information Management Requirements. Each of these areas represent a major segment of an automated IDEF4 tool's functionality and should be treated as equally important when developing an automated tool.

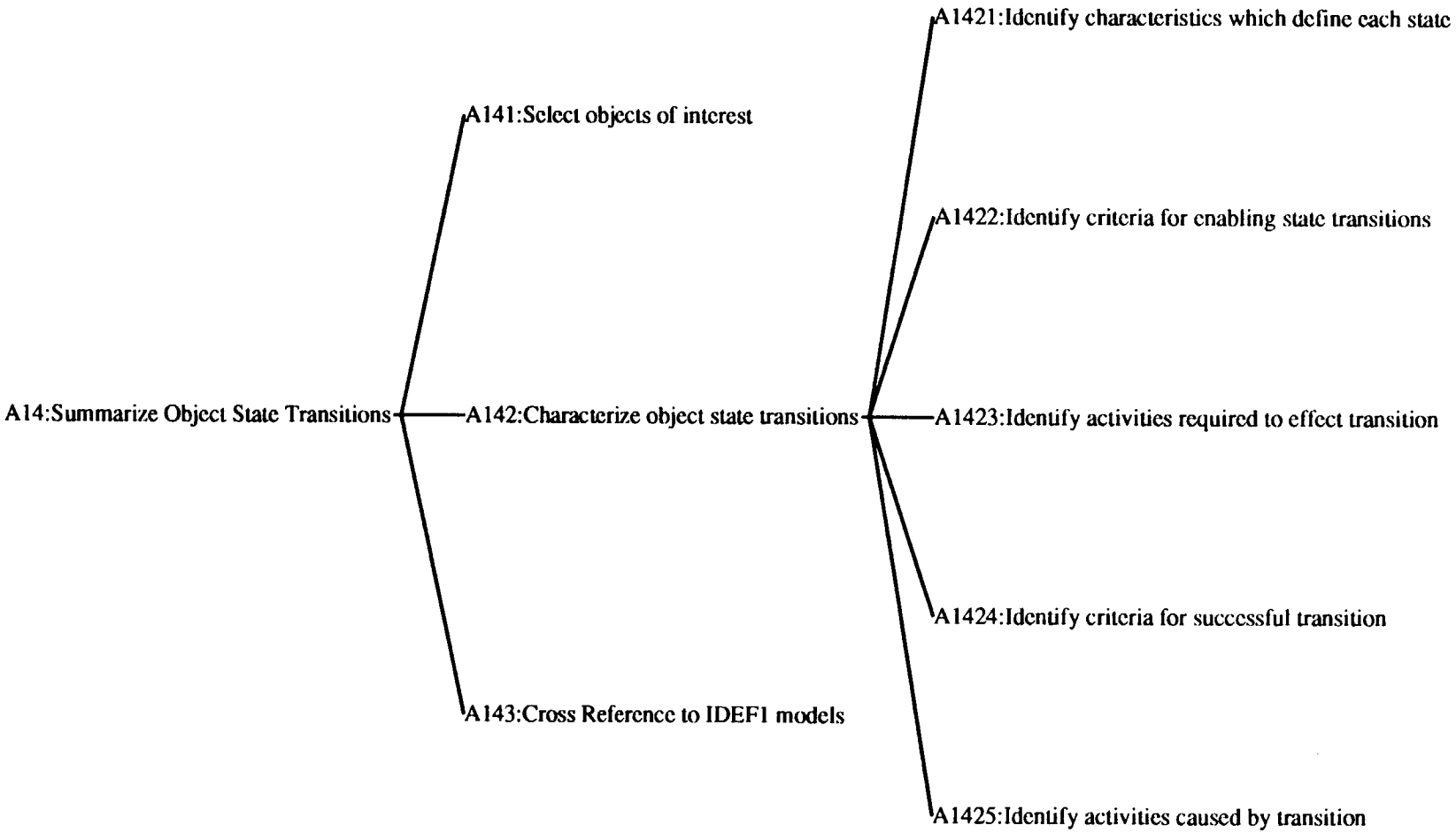
Appendix A: IDEFØ Model of IDEF3

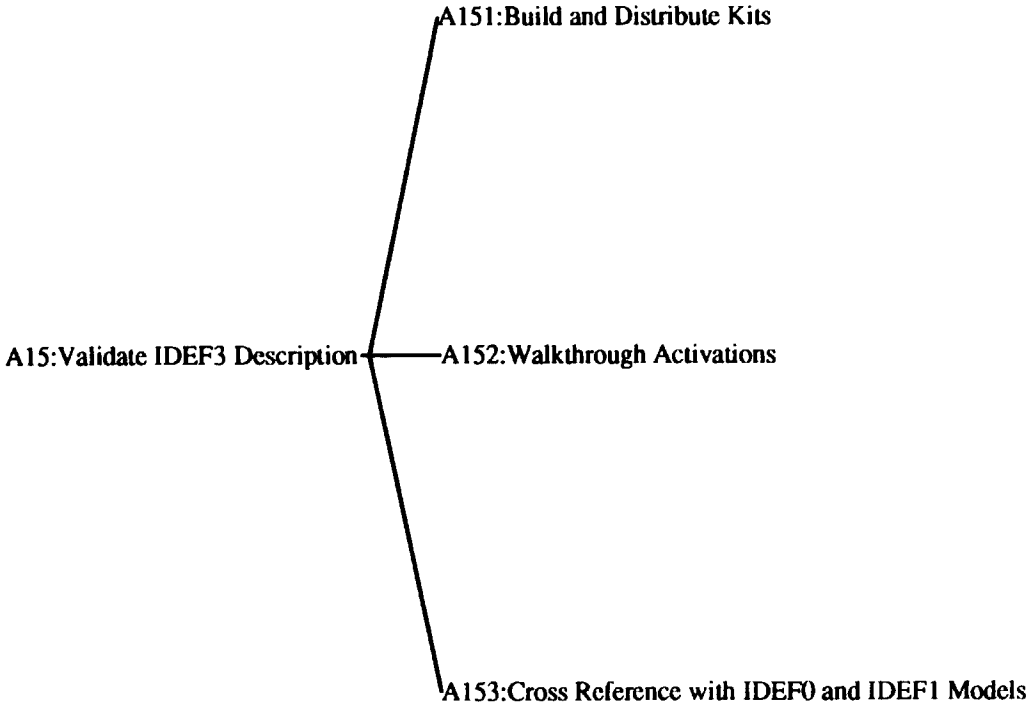

*:(root)
A0:Develop and Use IDEF3 Description
A1:Develop IDEF3 Description
A11:Bound the Context of the description, 2
A12:Collect Data, 3
A13:Formulate Process Flow Descriptions, 4
A14:Summarize Object State Transitions, 5
A15:Validate IDEF3 Description, 6
A2:Browse IDEF3 Description
A21:Load IDEF3 Description
A22:Select Scenario
A23:Navigate Process Network
A24:View Process Elaboration
A25:View Object State Transition Description
A3:Use IDEF3 Description
A31:Trace object, extracting selected processes
A32:Lasso set of processes for extraction
A33:Extract all processes related to a specified proce
A4:Maintain IDEF3 Description
A41:Maintain Process Flow Descriptions
A42:Maintain Object State Descriptions
A43:Maintain linkages to other models


A111:Define purpose for gathering description
A11:Bound the Context of the description
A112:Determine initial scope of description
A113:Identify major organizational scenarios


A121:Interview Area Experts
A122:Collect Names of Objects
A12:Collect Data
A123:Collect Names of Activities
A124:Collect Situation Descriptions


A131:Layout flows within each scenario/decomposition
A132:Add Junction Structures for Logic description
A13:Formulate Process Flow Descriptions
A133:Develop Elaborations for each UOB
A134:Develop Decompositions for selected UOBs

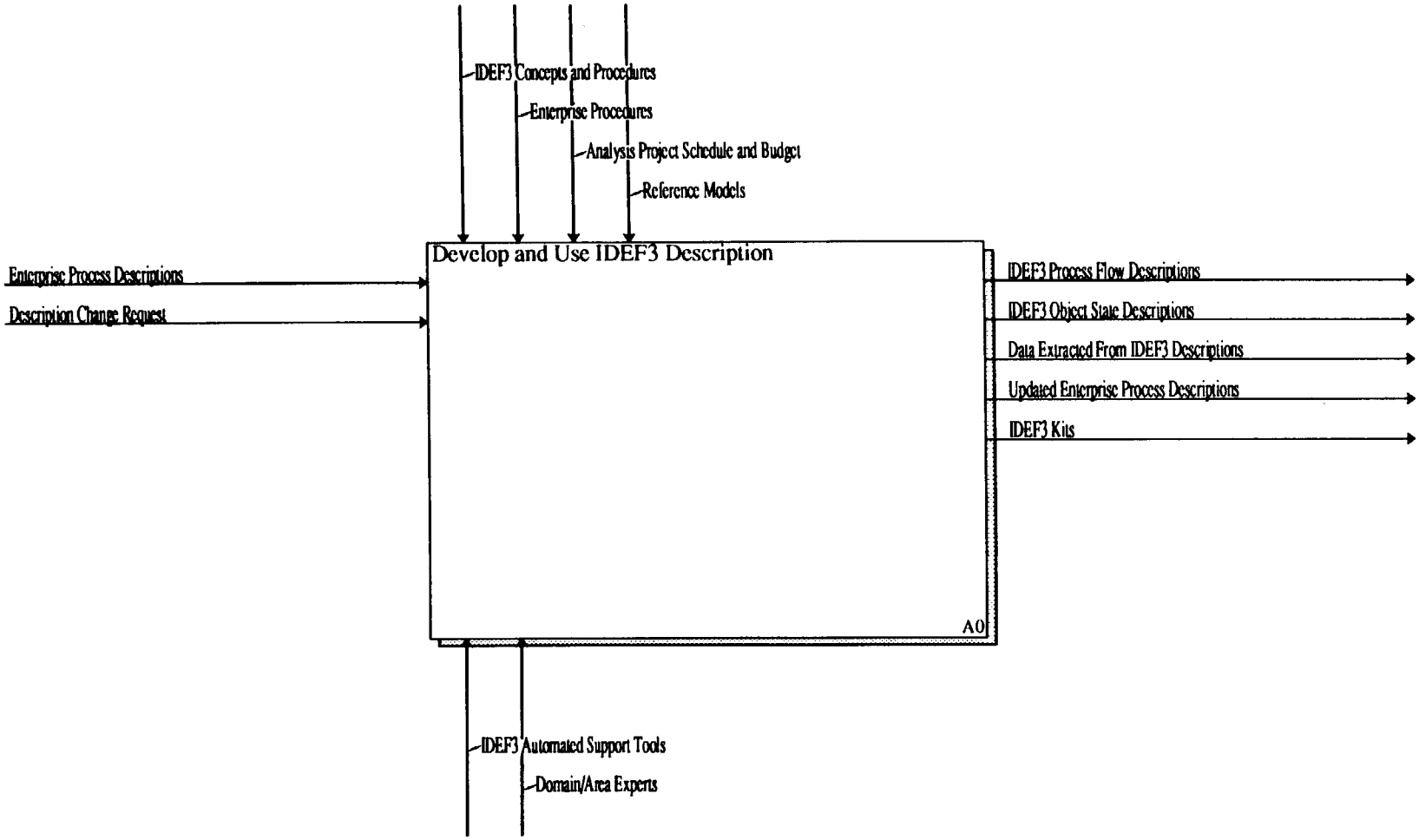

A14:Summarize Object State Transitions
A141:Select objects of interest
A142:Characterize object state transitions
A143:Cross Reference to IDEF1 models
A1421:Identify characteristics which define each state
A1422:Identify criteria for enabling state transitions
A1423:Identify activities required to effect transition
A1424:Identify criteria for successful transition
A1425:Identify activities caused by transition


A151:Build and Distribute Kits
A15:Validate IDEF3 Description
A152:Walkthrough Activations
A153:Cross Reference with IDEF0 and IDEF1 Models

Index

DIAGRAMS

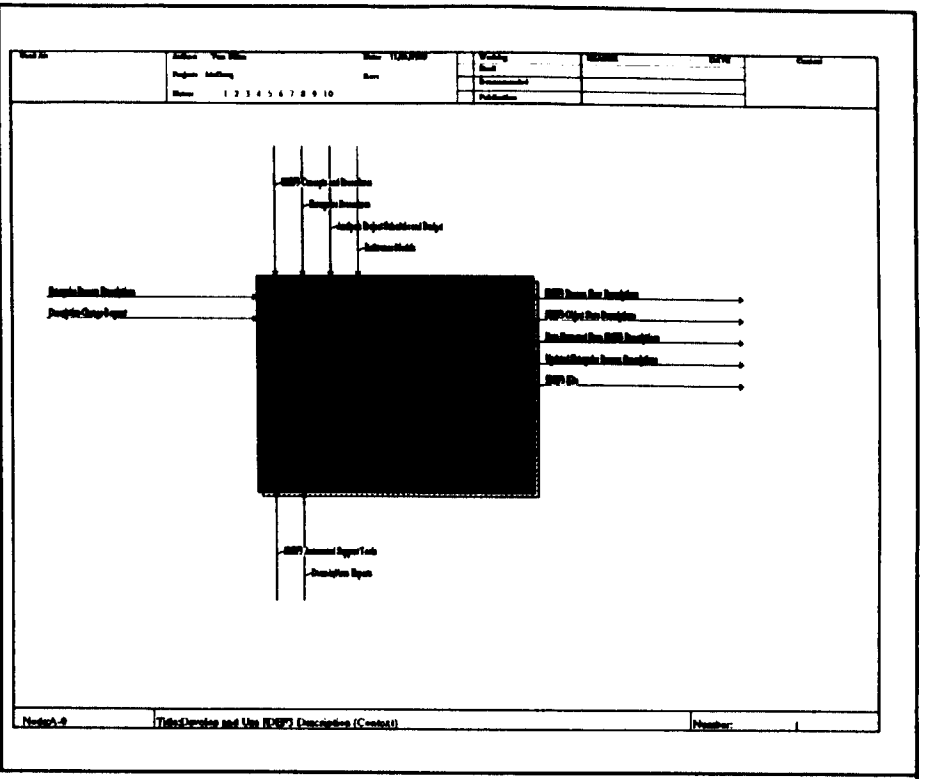
Used At:	Author: Tom Blinn	Date: 11/21/1989		Working	READER	DATE	Context
	Project: Idef3req	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



Node:A-0	Title:Develop and Use IDEF3 Description (Context)	Number: 9

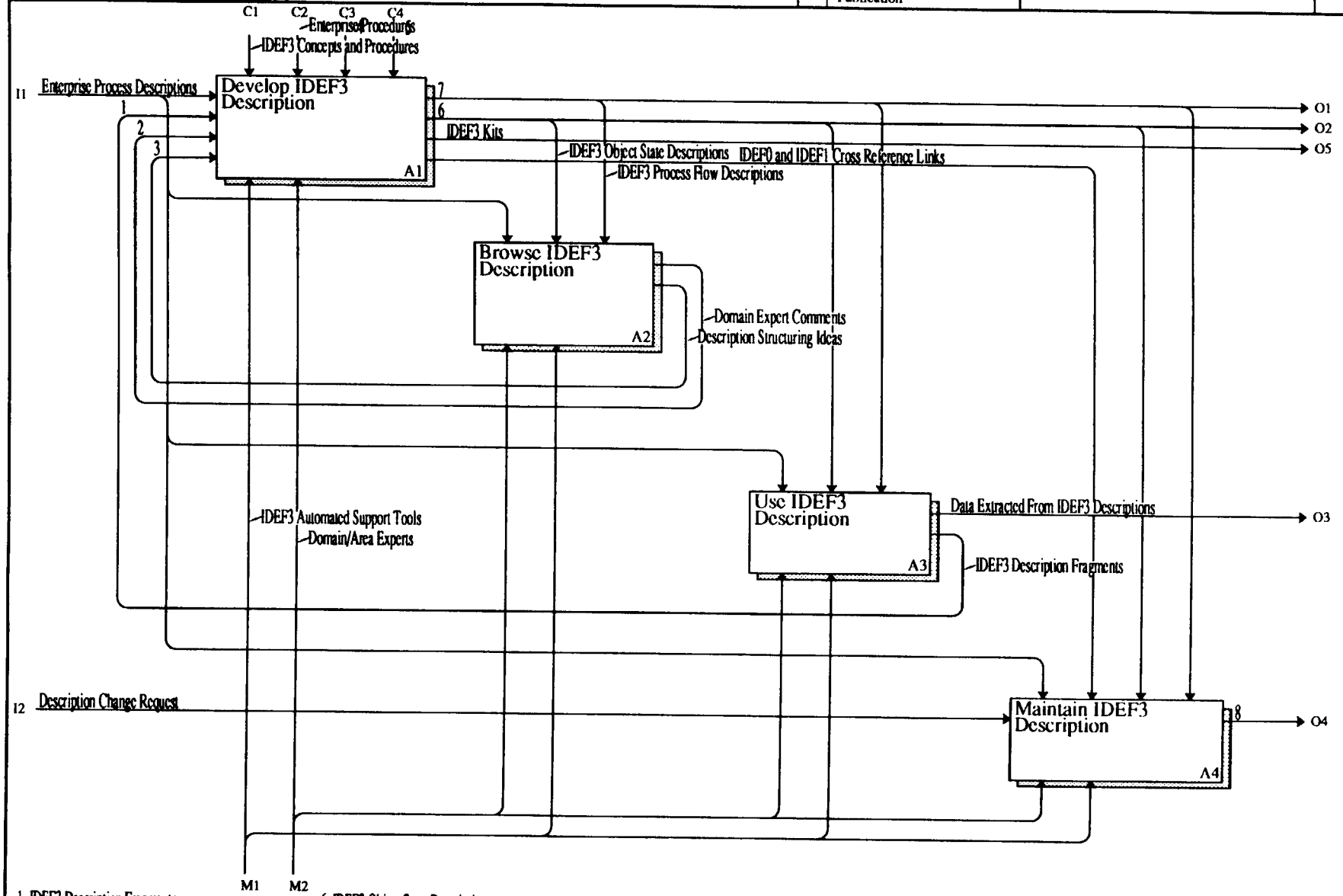
Develop and Use IDEF3 Description

Organize domain experts knowledge and understanding of how their system works in terms of processes, sequencing and logic, and object state transitions. Extract data from such descriptions to support system development descision making.



ORIGINAL PAGE IS
OF POOR QUALITY

Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef3req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

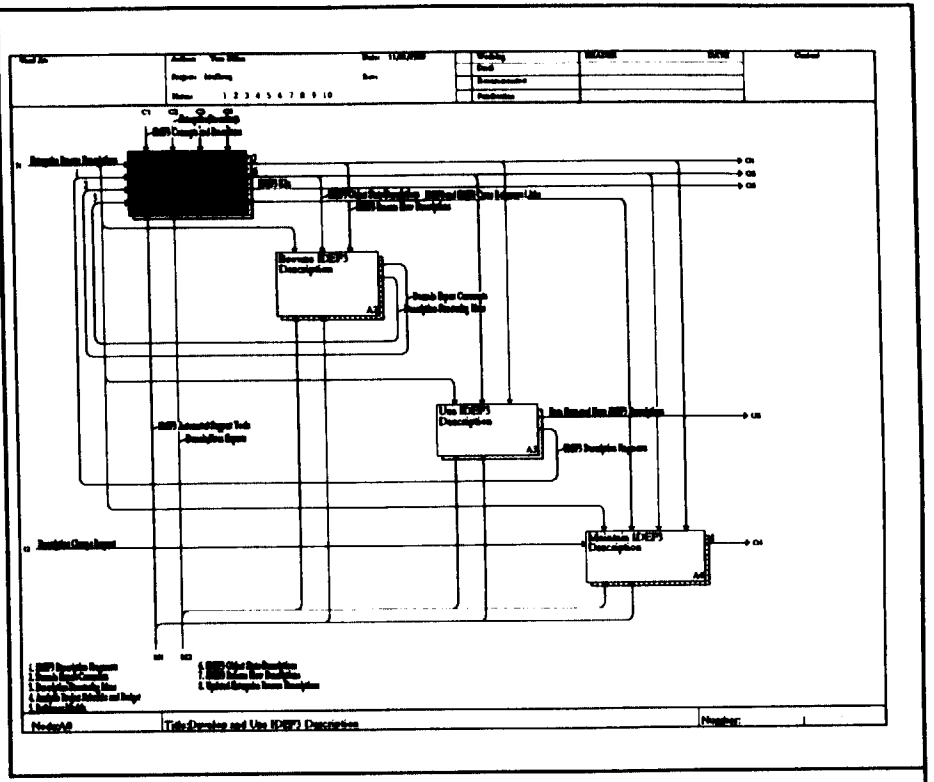


1. IDEF3 Description Fragments
2. Domain Expert Comments
3. Description Structuring Ideas
4. Analysis Project Schedule and Budget
5. Reference Models
6. IDEF3 Object State Descriptions
7. IDEF3 Process Flow Descriptions
8. Updated Enterprise Process Descriptions

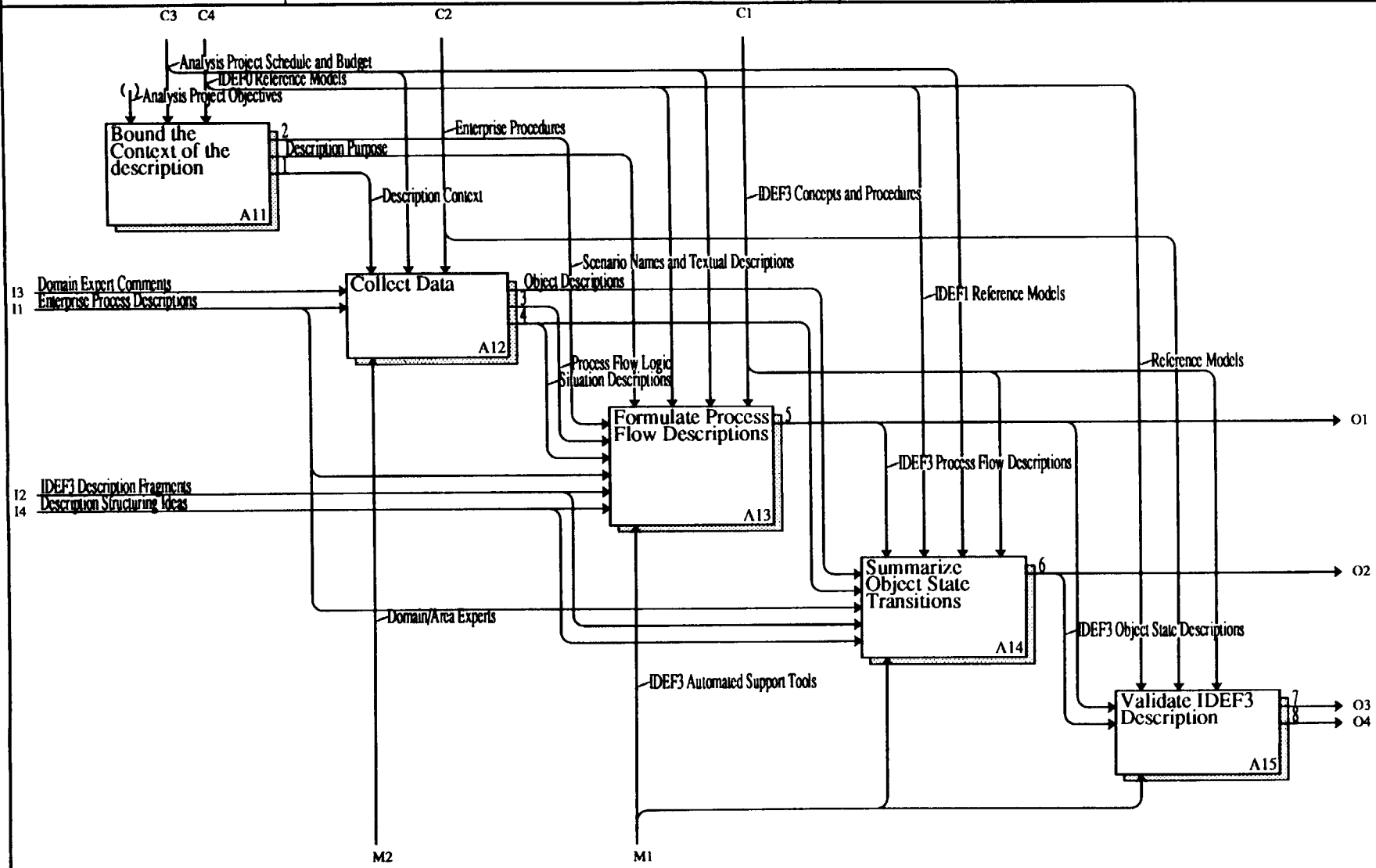
Node:A0	Title:Develop and Use IDEF3 Description	Number: 11

Develop IDEF3 Description

This activity includes the initial data gathering, description structuring and validation activities associated with an IDEF3 description authoring process.



Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef3req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

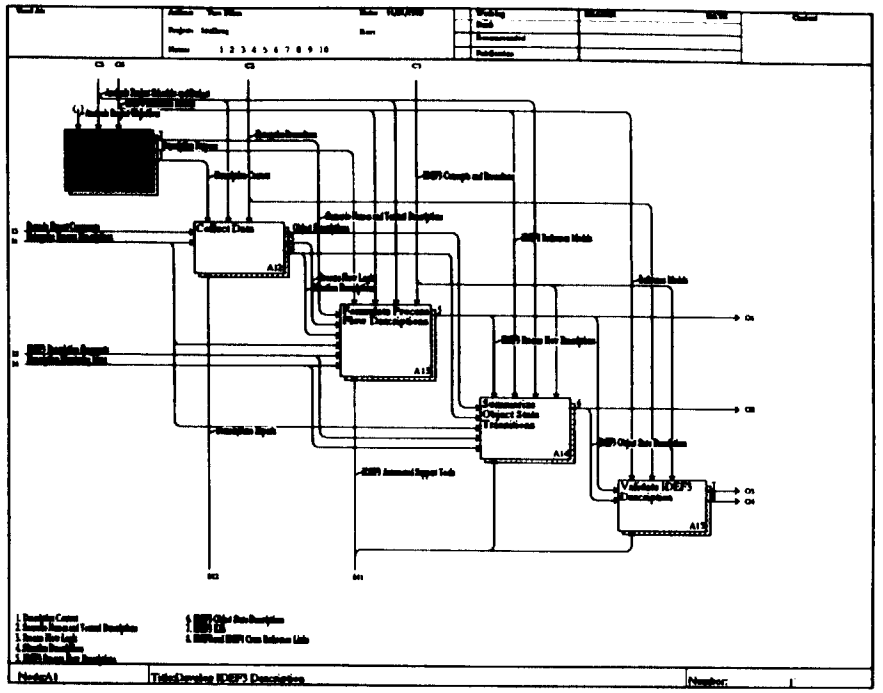


1. Description Context
2. Scenario Names and Textual Descriptions
3. Process Flow Logic
4. Situation Descriptions
5. IDEF3 Process Flow Descriptions
6. IDEF3 Object State Descriptions
7. IDEF3 Kits
8. IDEF0 and IDEF1 Cross Reference Links

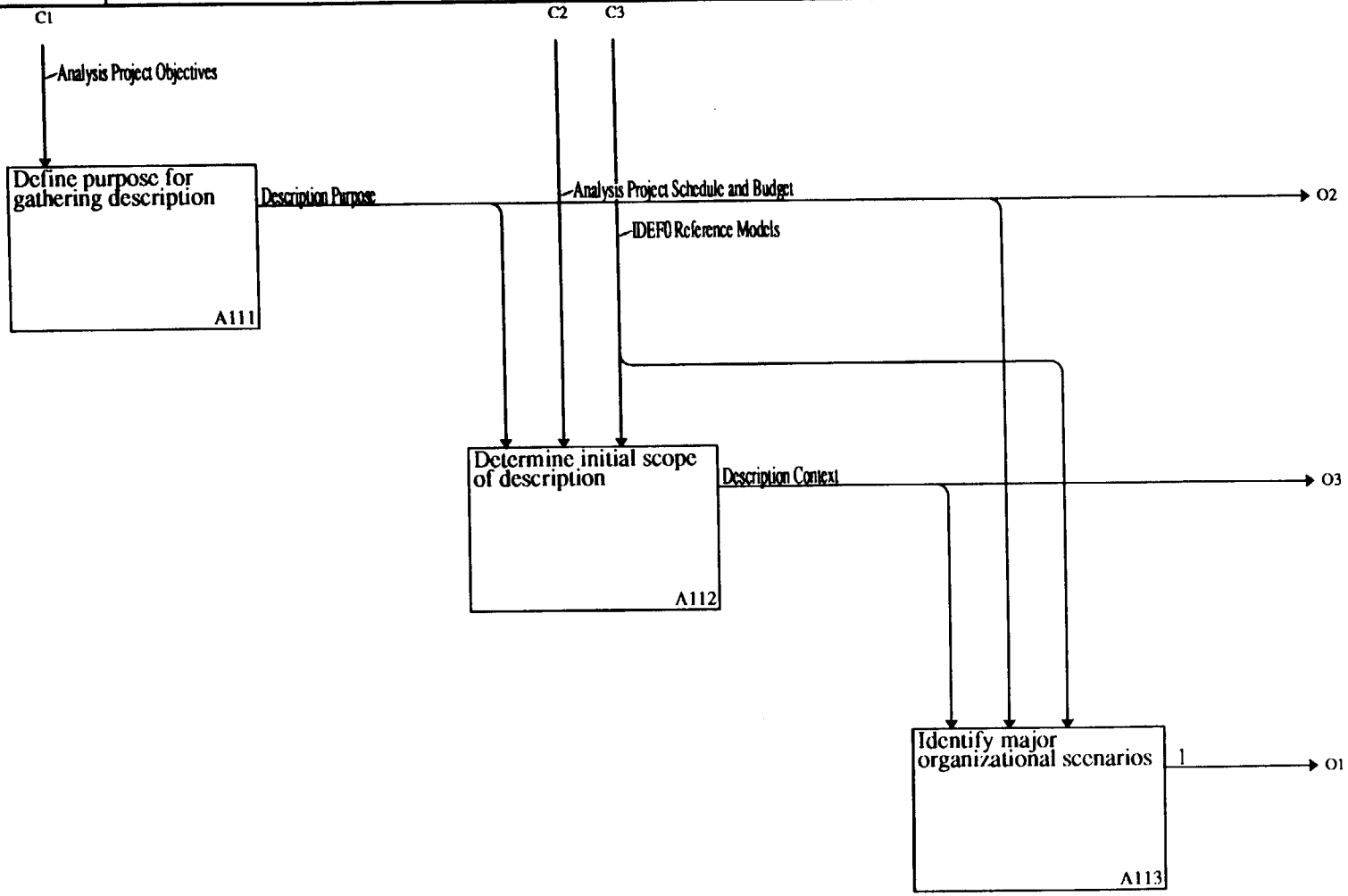
Node:A1	Title:Develop IDEF3 Description	Number: 13

Bound the Context of the description

Like IDEF0, IDEF1, and even IDEF1x the scope and purpose of an IDEF3 model must be determined in order to give focus to the data collection process. Unlike IDEF0 however, the context of IDEF3 can be easily expanded even after the description is fairly well along. This is primarily due to the ability to easily include other scenarios and decomposition views.



Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef3req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

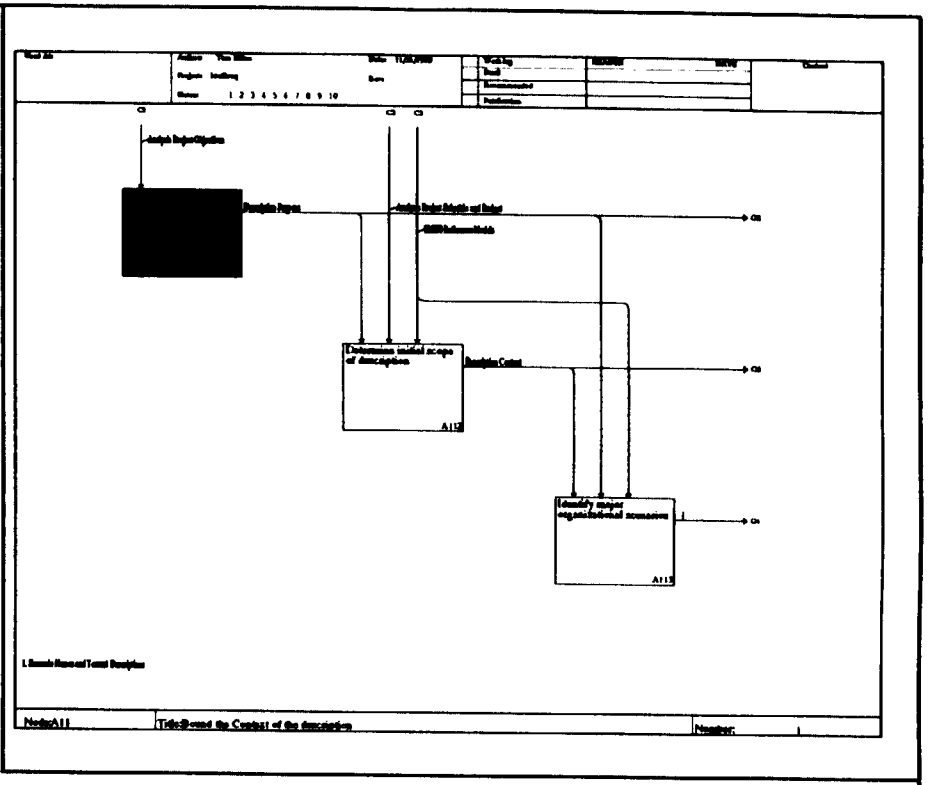


1. Scenario Names and Textual Descriptions

Node:A11	Title:Bound the Context of the description	Number: 15

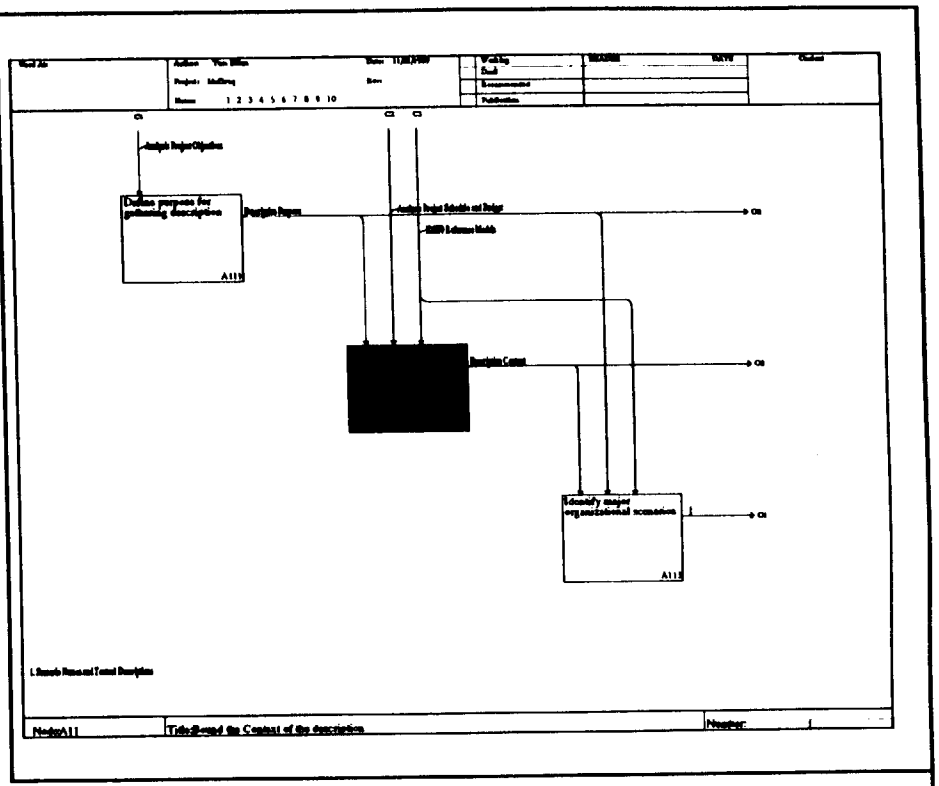
Define purpose for gathering description

This activity defines the goals for the process description development. It outlines what should be accomplished by performing this detailed analysis of the enterprise process.



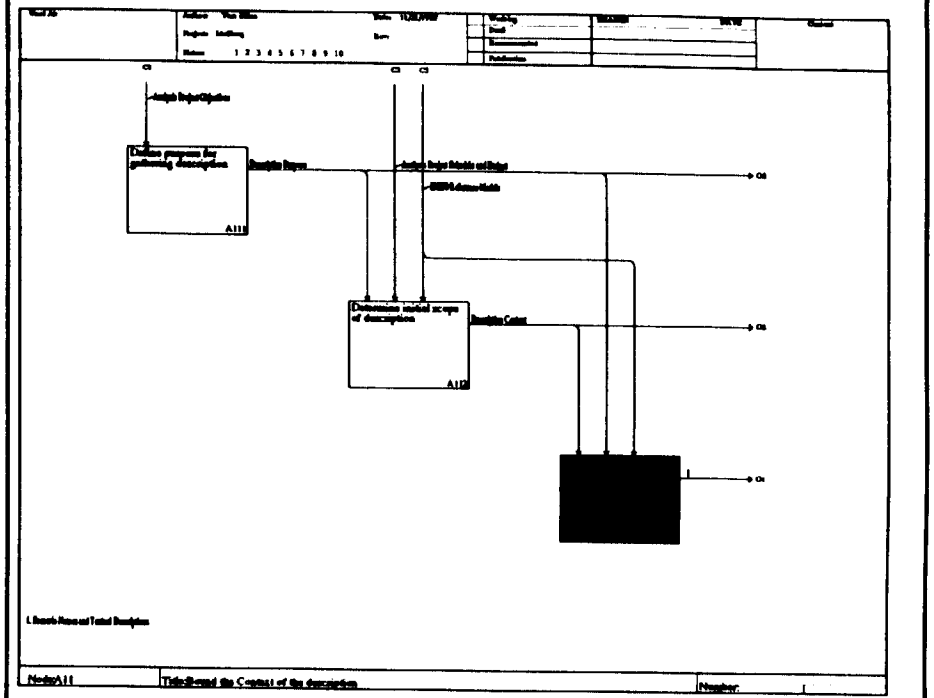
Determine initial scope of description

At this point, a decision must be made as to the extent of the analysis of the enterprise process. This decision will provide the scope of the process description development.



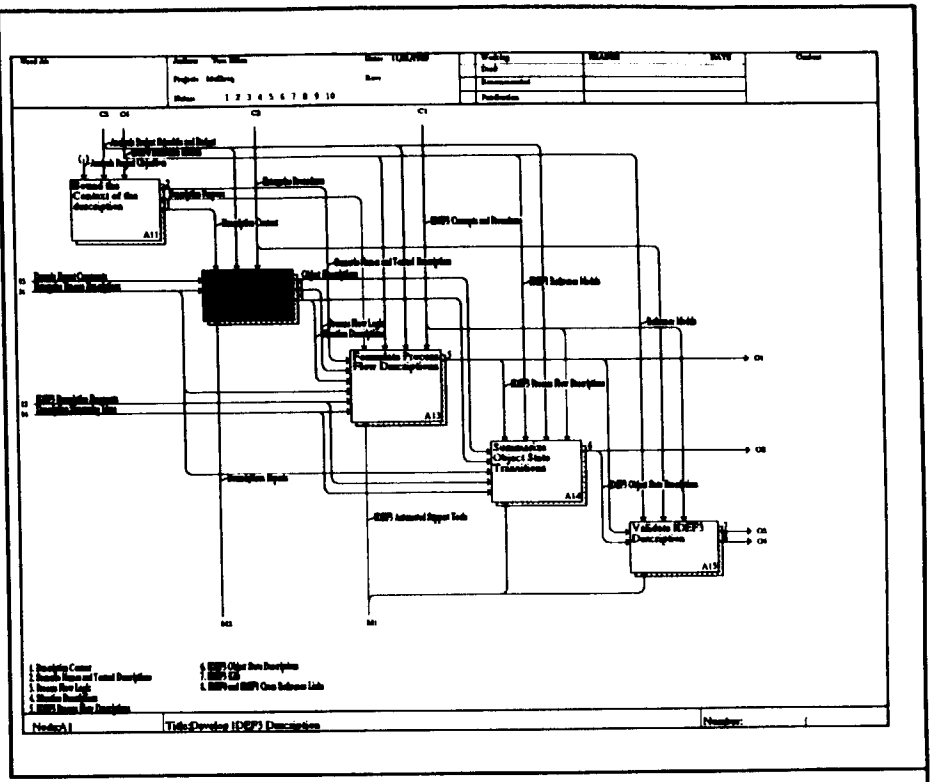
Identify major organizational scenarios

This activity identifies the various perspectives that should be included in the process description.



Collect Data

As with any model development, accurate data must be collected before the development of a process description can begin. This can be the most important phase of the model development. Without a clear understanding of the enterprise process, an accurate process model is impossible. It is imperative that the information collected at this stage be useful and pertinent.



Used At:

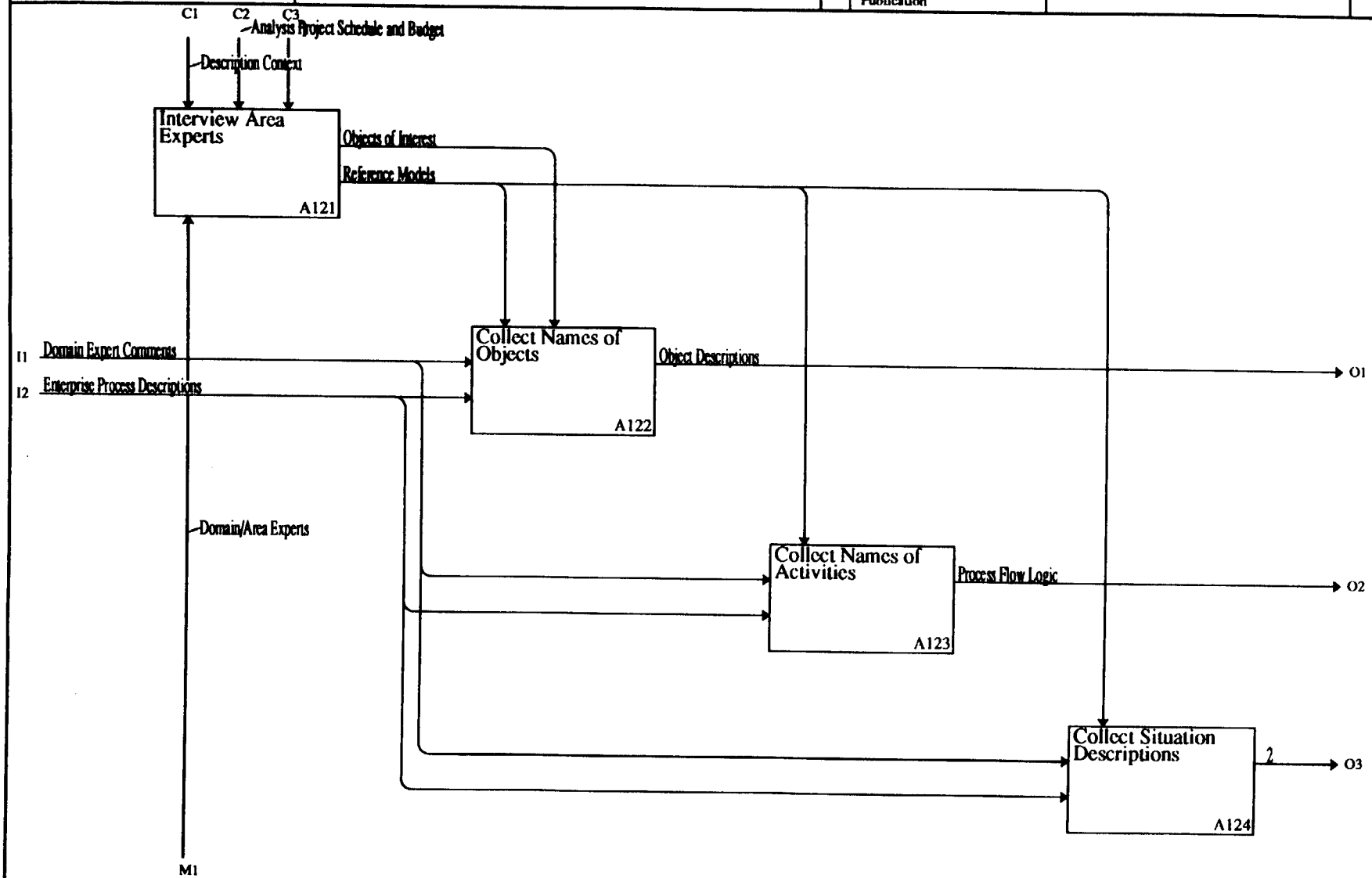
Author: Tom Blinn Date: 11/21/1989

Project: Idef3req Rev:

Notes: 1 2 3 4 5 6 7 8 9 10

	Working	READER	DATE
	Draft		
	Recommended		
	Publication		

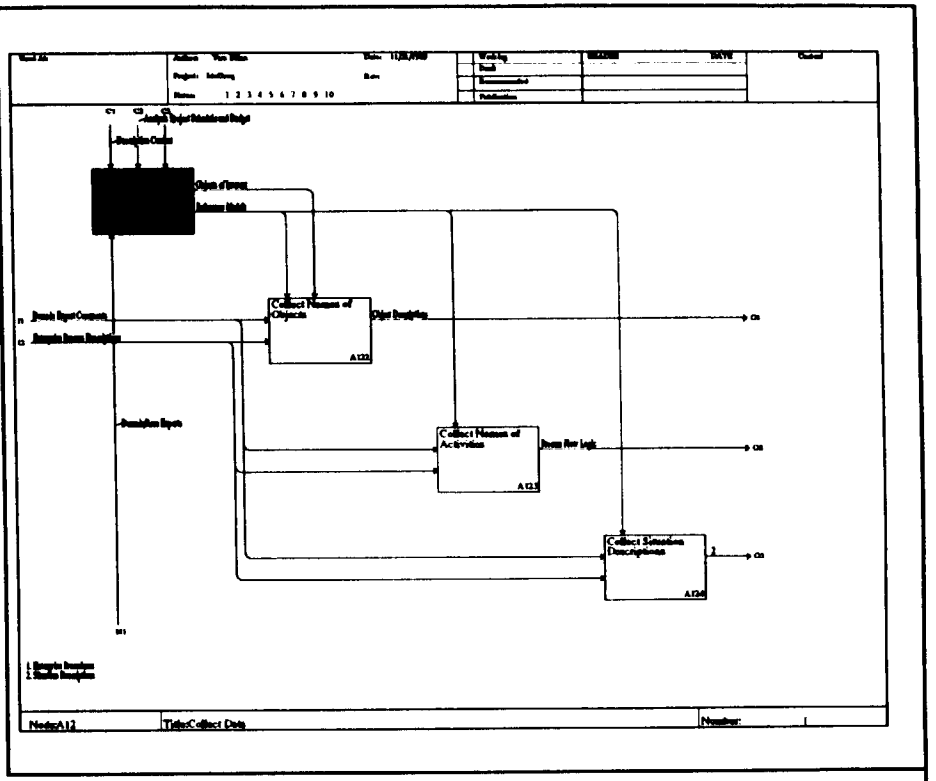
Context



Node:A12 Title:Collect Data Number: 20

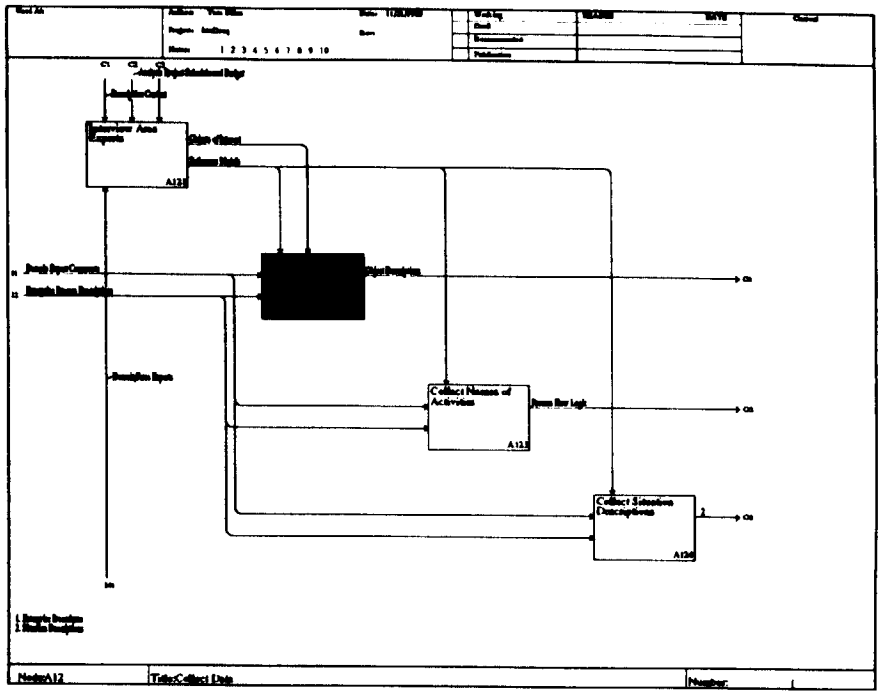
Interview Area Experts

The most valuable information about an enterprise is held by the domain experts. Therefore, it is critical that these experts be interviewed to acquire as much of their knowledge of the enterprise as possible.



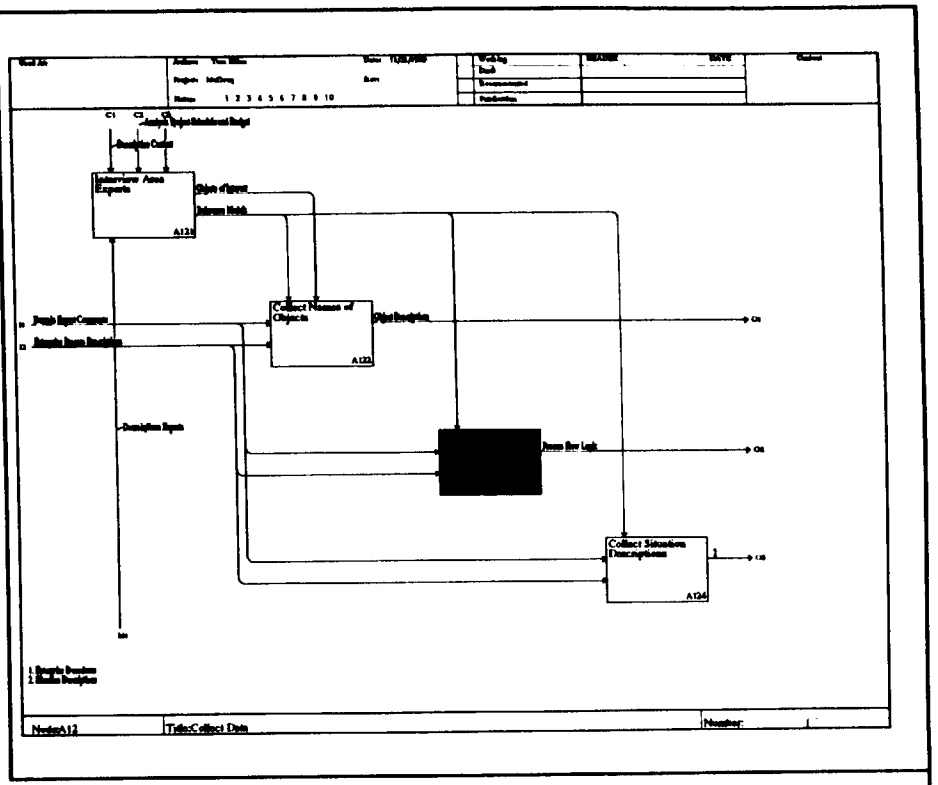
Collect Names of Objects

Here, the modeler should collect the names of objects that might play a part in the process description.



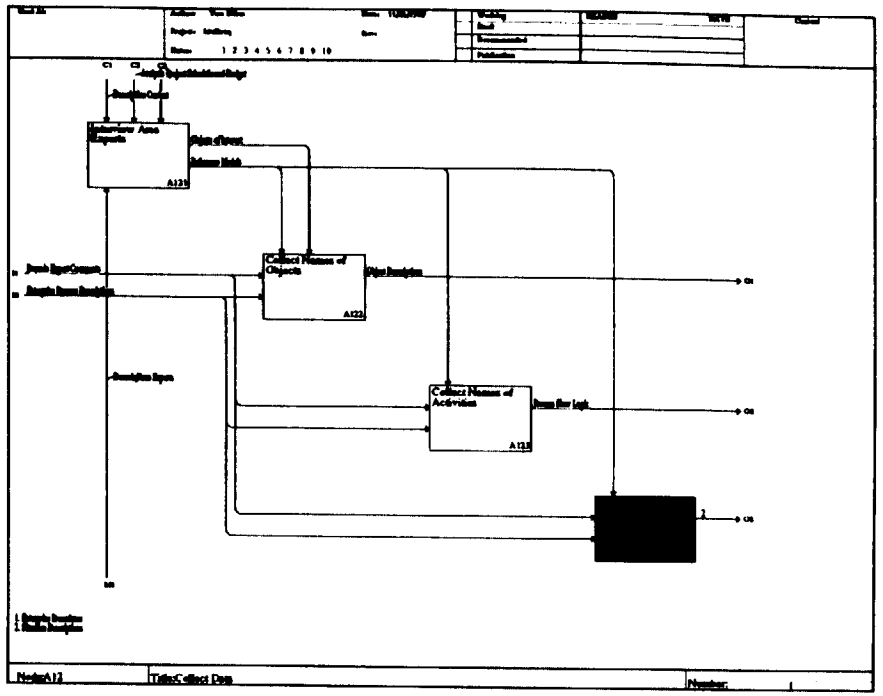
Collect Names of Activities

The modeler should collect names of activities that are performed in the execution of an enterprise process.



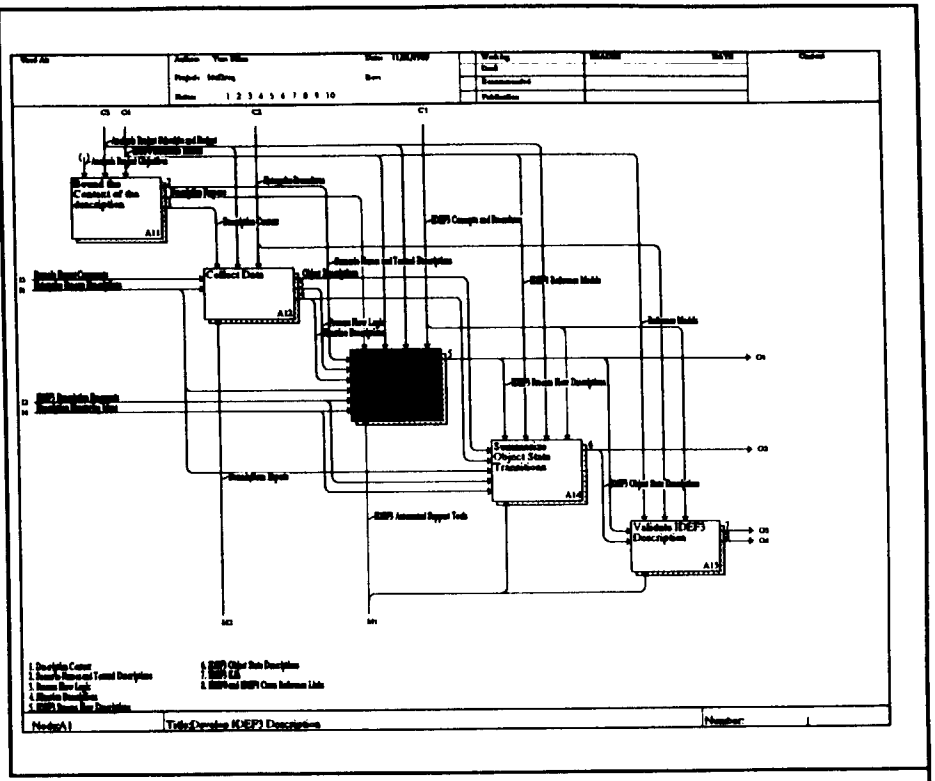
Collect Situation Descriptions

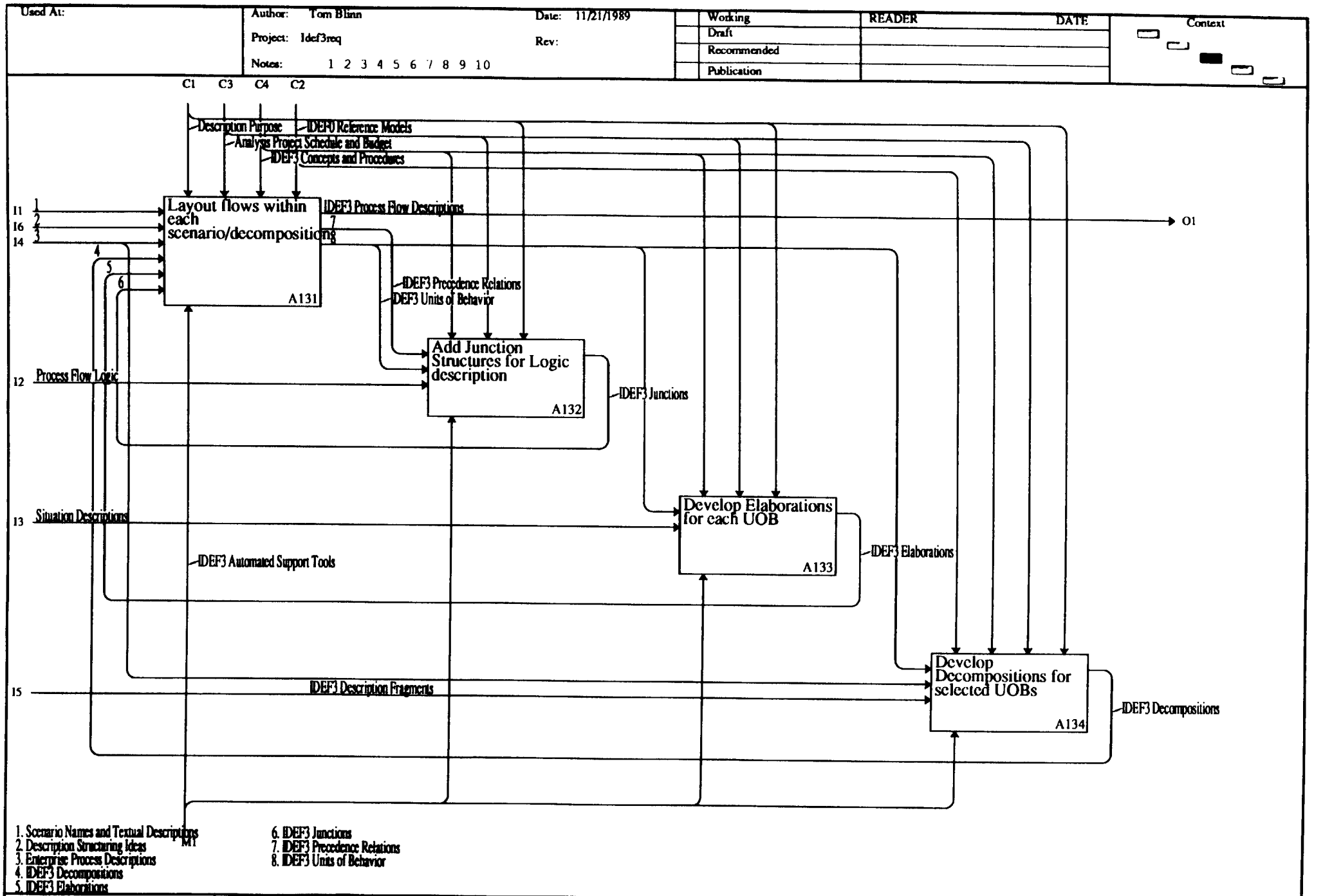
The modeler should collect descriptions of various situations that may occur within the execution of an enterprise process.



Formulate Process Flow Descriptions

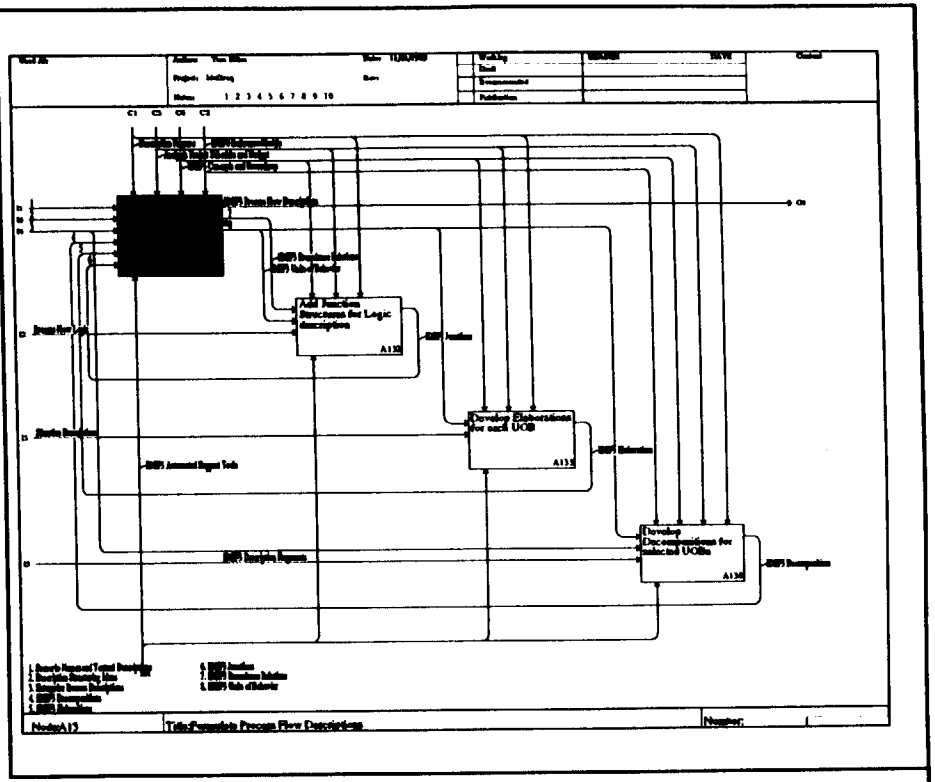
This activity produces the actual process flow descriptions of the enterprise process. The process flow is laid out as well as the precedence and logical relations. This acitivity also includes the development of the UOB elaboration and decompositions.




Used At:
Author: Tom Blinn
Date: 11/21/1989
Project: Idef3req
Rev:
Notes: 1 2 3 4 5 6 7 8 9 10
Working
Draft
Recommended
Publication
READER
DATE
Context
C1
C3
C4
C2
Description Purpose
IDEF0 Reference Models
Analysis Project Schedule and Budget
IDEF3 Concepts and Procedures
I1
I6
I4
Layout flows within each scenario/decomposition
A131
IDEF3 Process Flow Descriptions
O1
IDEF3 Precedence Relations
IDEF3 Units of Behavior
Add Junction Structures for Logic description
A132
I2
Process Flow Logic
IDEF3 Junctions
Develop Elaborations for each UOB
A133
I3
Situation Descriptions
IDEF3 Elaborations
IDEF3 Automated Support Tools
Develop Decompositions for selected UOBs
A134
I5
IDEF3 Description Fragments
IDEF3 Decompositions
M1
1. Scenario Names and Textual Descriptions
2. Description Structuring Ideas
3. Enterprise Process Descriptions
4. IDEF3 Decompositions
5. IDEF3 Elaborations
6. IDEF3 Junctions
7. IDEF3 Precedence Relations
8. IDEF3 Units of Behavior
Node:A13
Title:Formulate Process Flow Descriptions
Number: 26

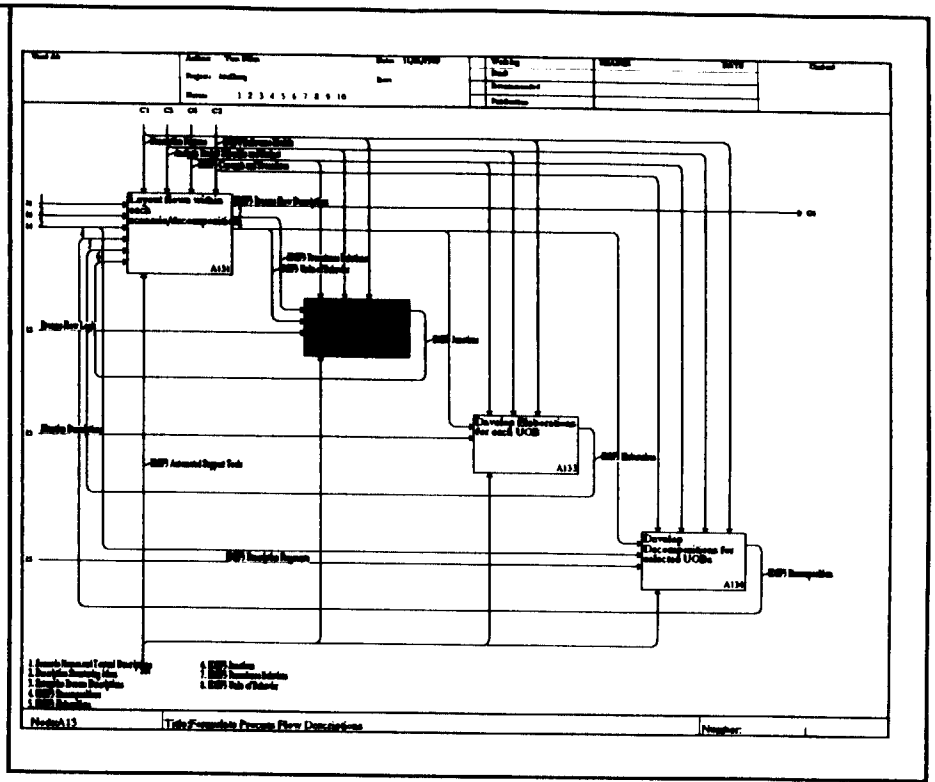
Layout flows within each scenario/decomposition

This activity involves defining the Units of Behavior (UOB) that are involved in a scenario. The flows between the UOBs are also specified.



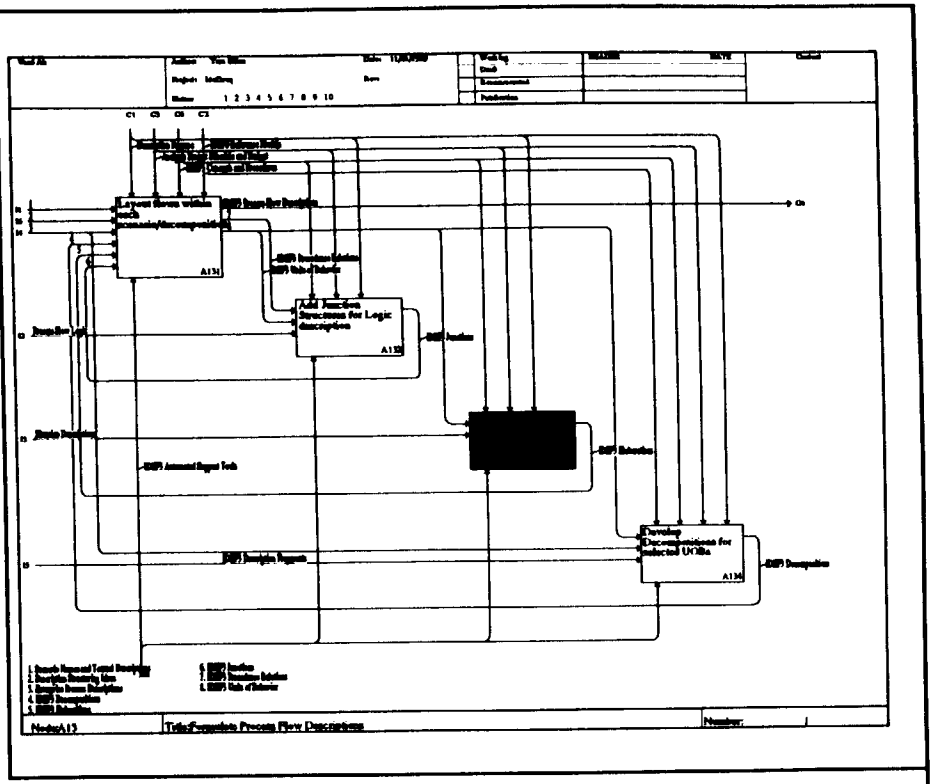
Add Junction Structures for Logic description

After the precedence relationships have been defined, it is necessary to specify the synchronization requirements for the branches in the process flow. During this activity, the junction structures should be defined.



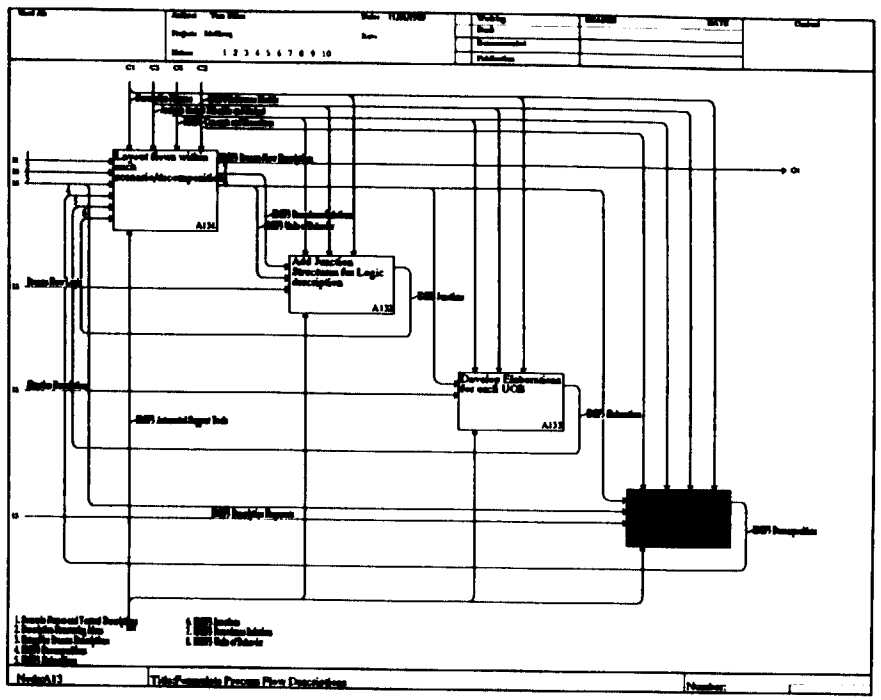
Develop Elaborations for each UOB

This activity defines the objects that are involved or somehow related to the UOB. The objects can be tagged as an "agent", as a "participant", as "affected", as "created", or as "deleted" depending on their purpose in the UOB.



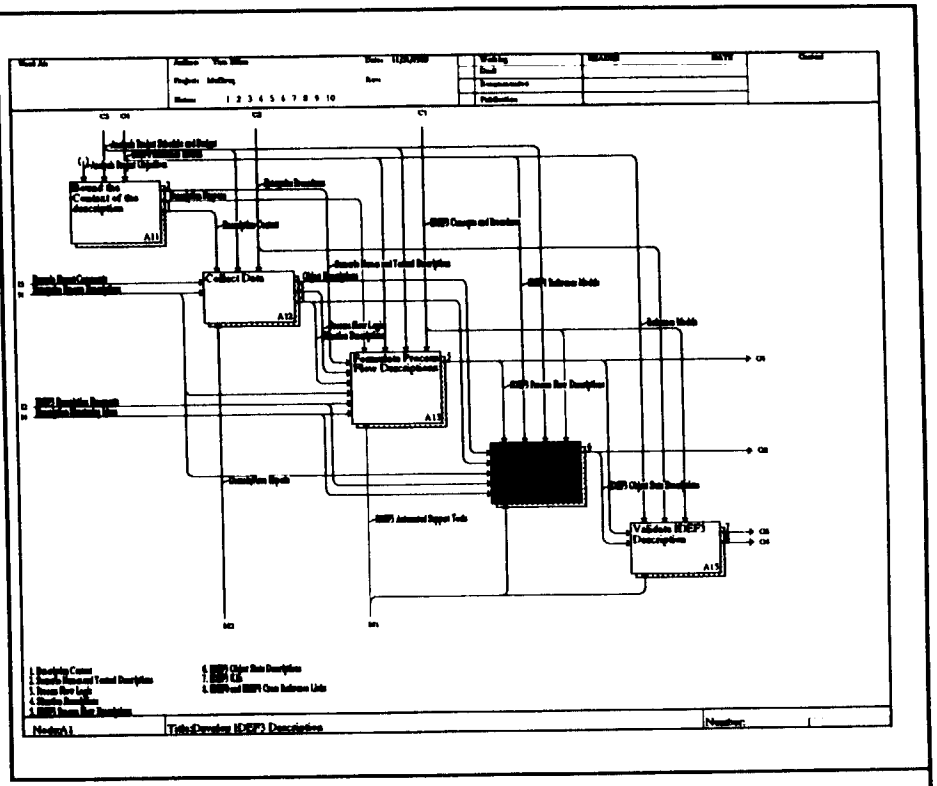
Develop Decompositions for selected UOBs

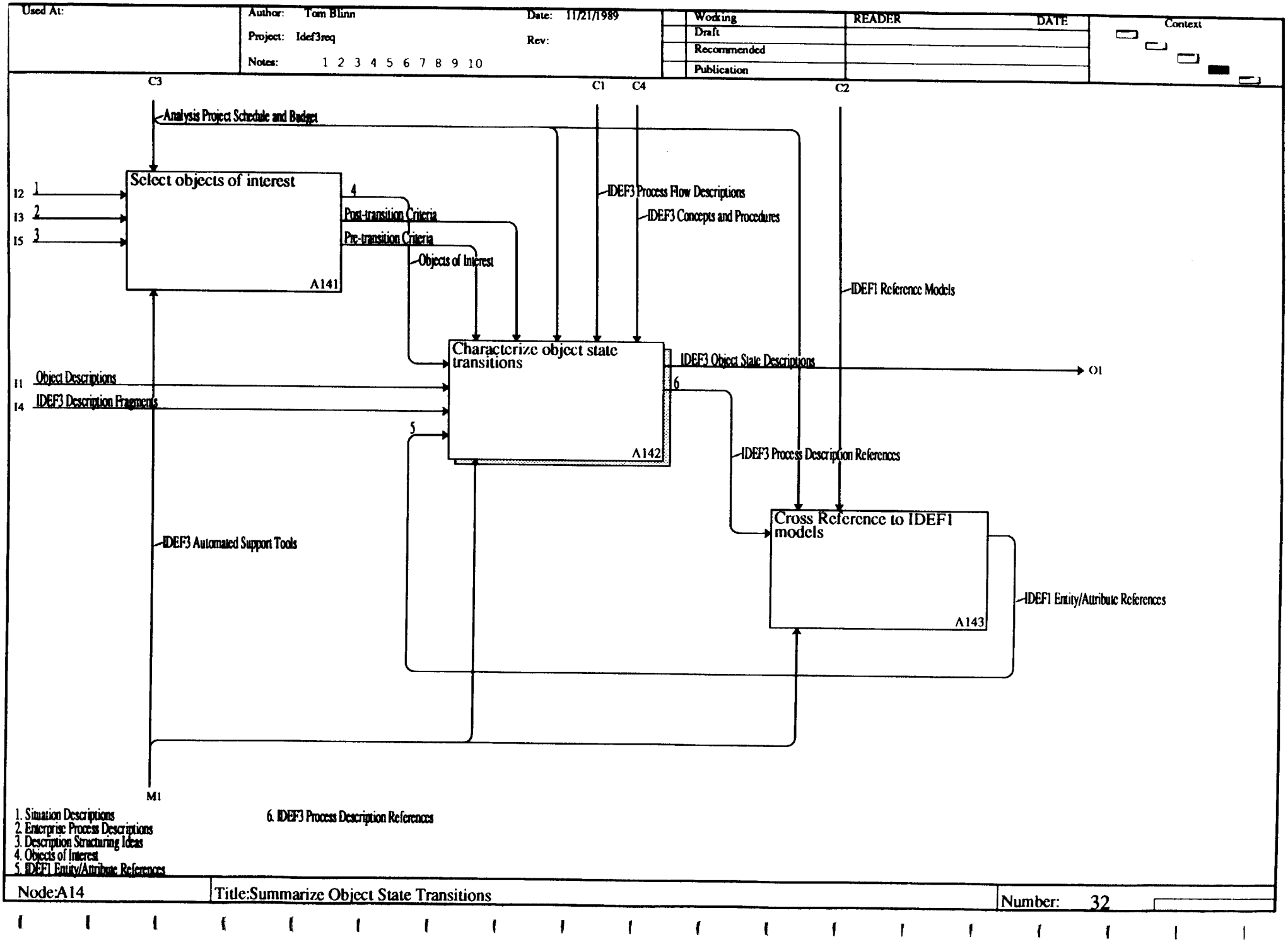
It is at this point that more detailed descriptions of a UOB can be developed. These decompositions are themselves process descriptions made up of other UOBs, links, and junctions that give a more detailed description of the parent UOB.



Summarize Object State Transitions

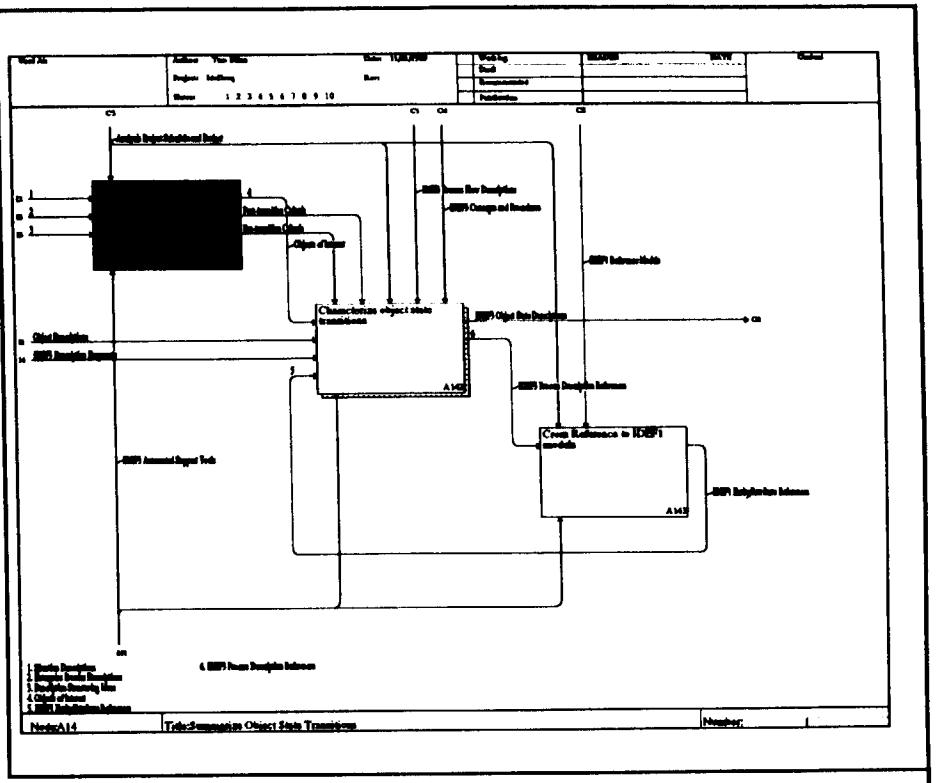
The object state transitions describe the changes that objects related to processes undergo during the execution of that process. These descriptions indicate the conditions that must be met before a transition can begin or can complete. Often, these conditions prevent the activation of processes.




Used At:
Author: Tom Blinn
Date: 11/21/1989
Project: Idef3req
Rev:
Notes: 1 2 3 4 5 6 7 8 9 10
Working
Draft
Recommended
Publication
READER
DATE
Context
C3
C1
C4
C2
Analysis Project Schedule and Budget
Select objects of interest
A141
I2 1
I3 2
I5 3
4
Post-transition Criteria
Pre-transition Criteria
Objects of Interest
IDEF3 Process Flow Descriptions
IDEF3 Concepts and Procedures
IDEF1 Reference Models
Characterize object state transitions
A142
I1 Object Descriptions
I4 IDEF3 Description Fragments
5
IDEF3 Object State Descriptions
O1
6
IDEF3 Process Description References
Cross Reference to IDEF1 models
A143
IDEF1 Entity/Attribute References
IDEF3 Automated Support Tools
M1
1. Situation Descriptions
2. Enterprise Process Descriptions
3. Description Structuring Ideas
4. Objects of Interest
5. IDEF1 Entity/Attribute References
6. IDEF3 Process Description References
Node:A14
Title:Summarize Object State Transitions
Number: 32

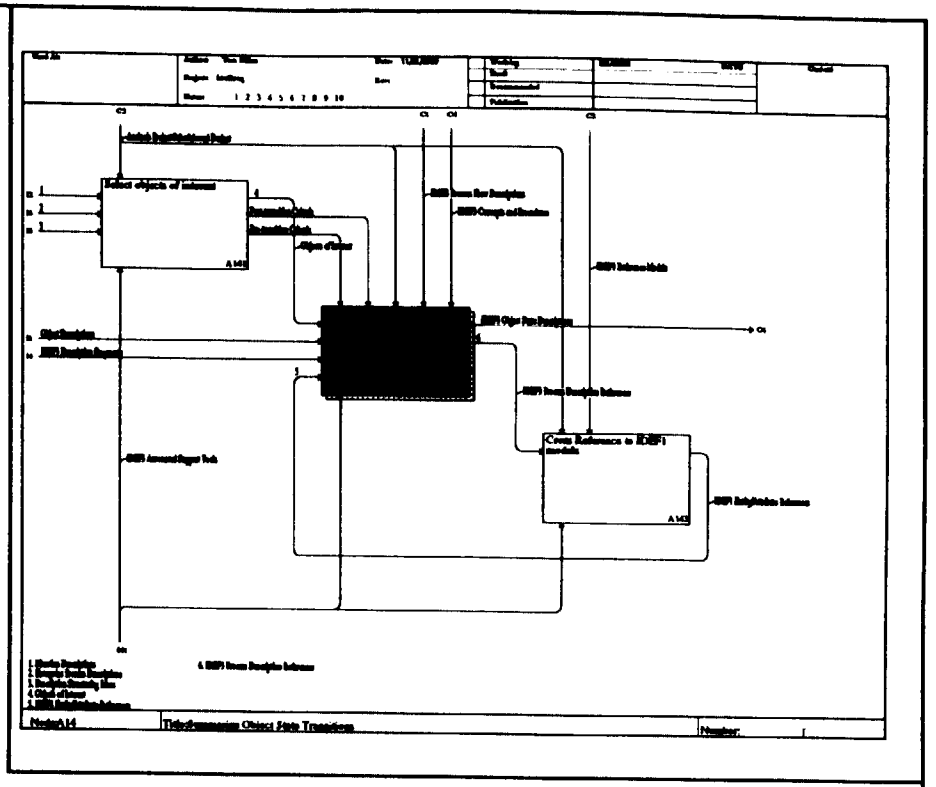
Select objects of interest

At this point, the modeler must select those objects that are most interesting to the process description. Some objects within a process may not be significant to the current scenario and do not warrant a state transition description.

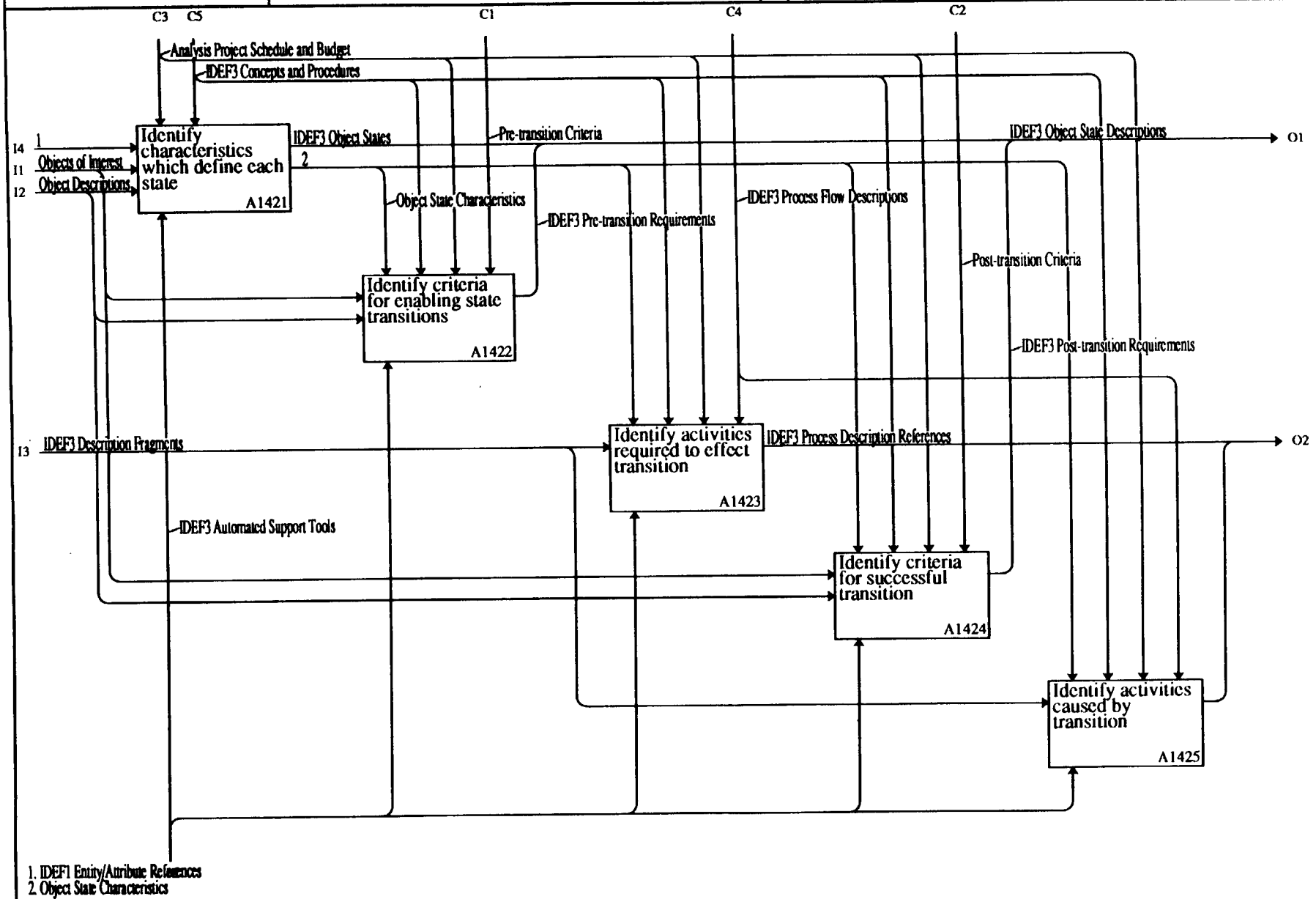


Characterize object state transitions

This activity involves the definition of an object state transition description.



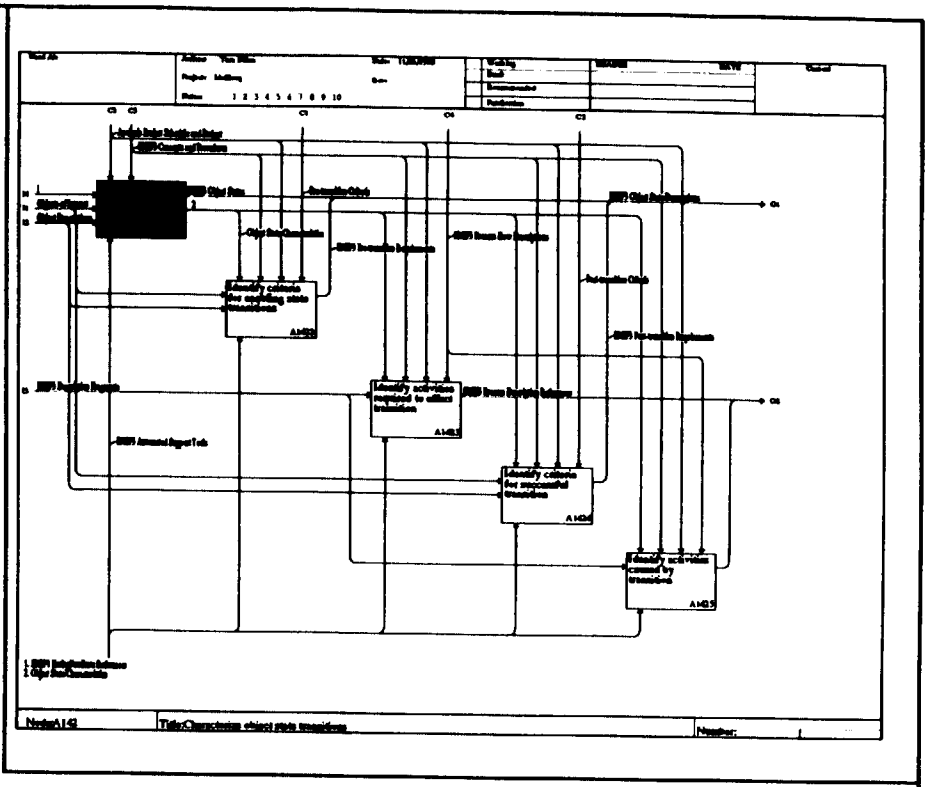
Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef3req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



Node:A142	Title:Characterize object state transitions	Number: 35

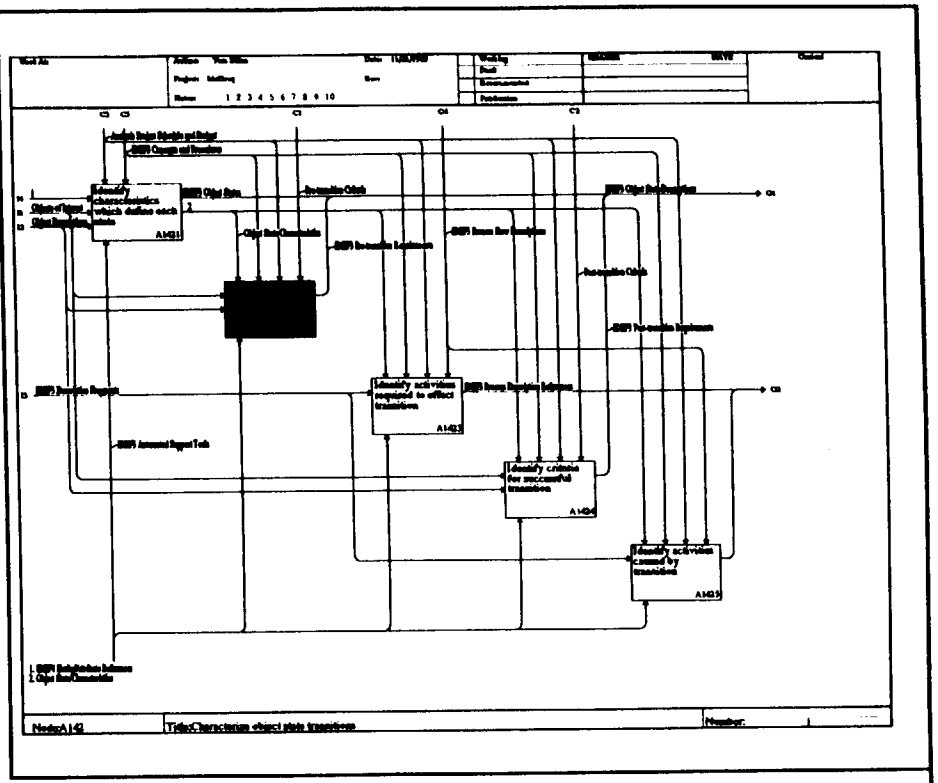
Identify characteristics which define each state

In essence, this activity defines an object state. The object involved, as well as constraints that specify the state of the object are defined.



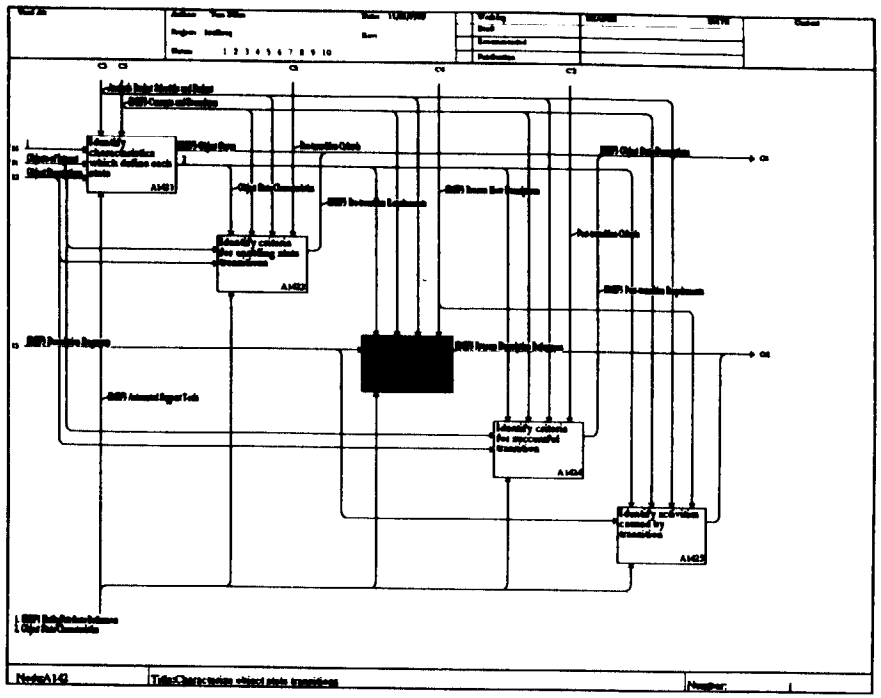
Identify criteria for enabling state transitions

Identify the preconditions which must be satisfied before an object state transition is possible (necessary conditions).



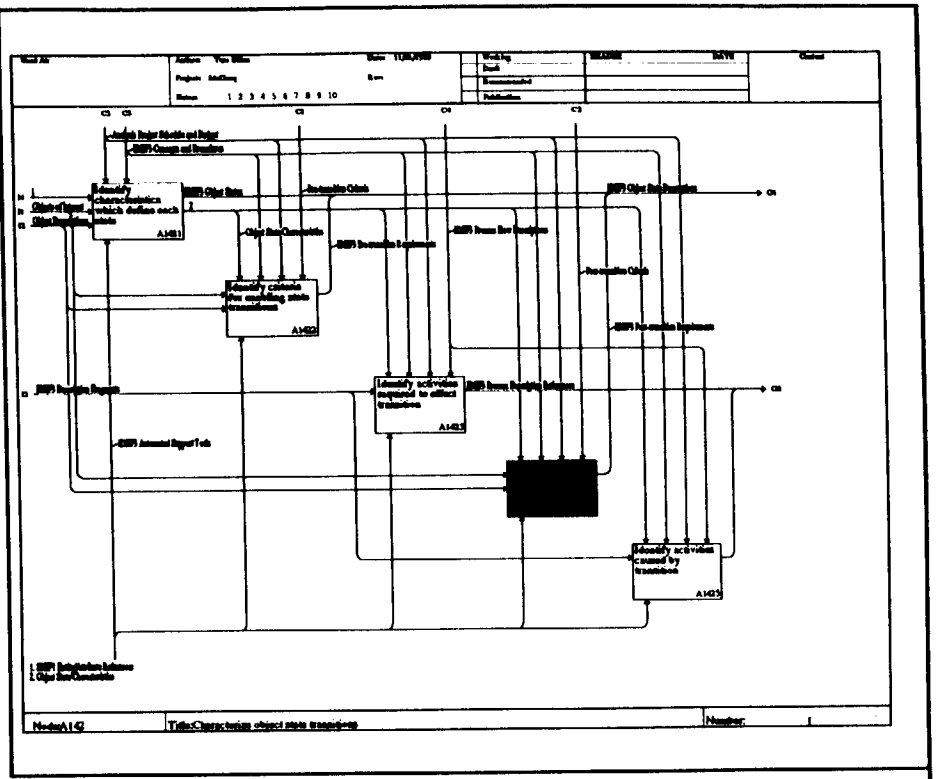
Identify activities required to effect transition

This activity references the UOBs that must be activated before this state transition can be considered for effection.



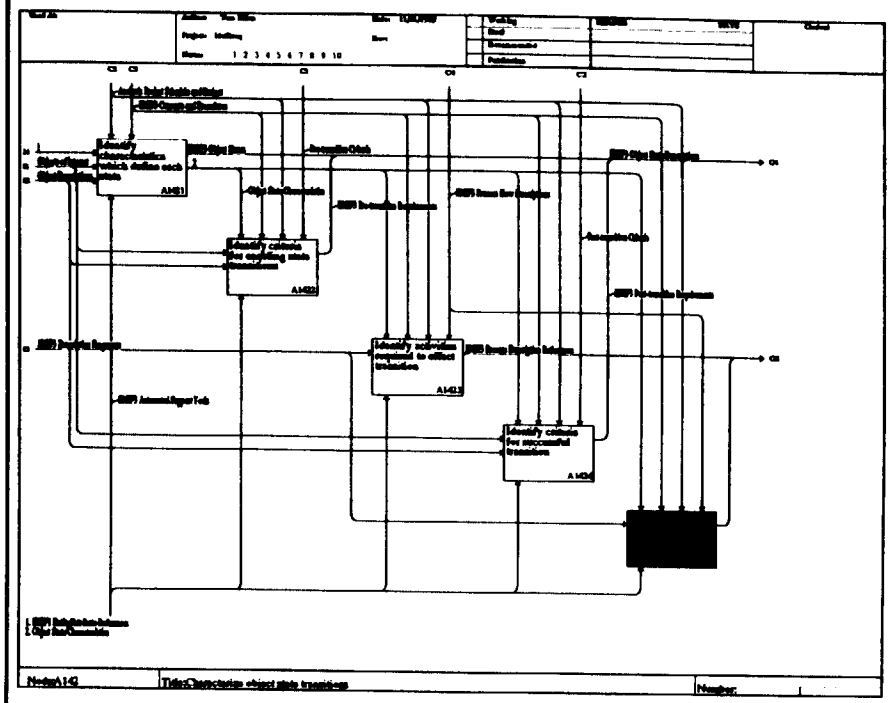
Identify criteria for successful transition

Criteria for an object to successfully complete a transition.



Identify activities caused by transition

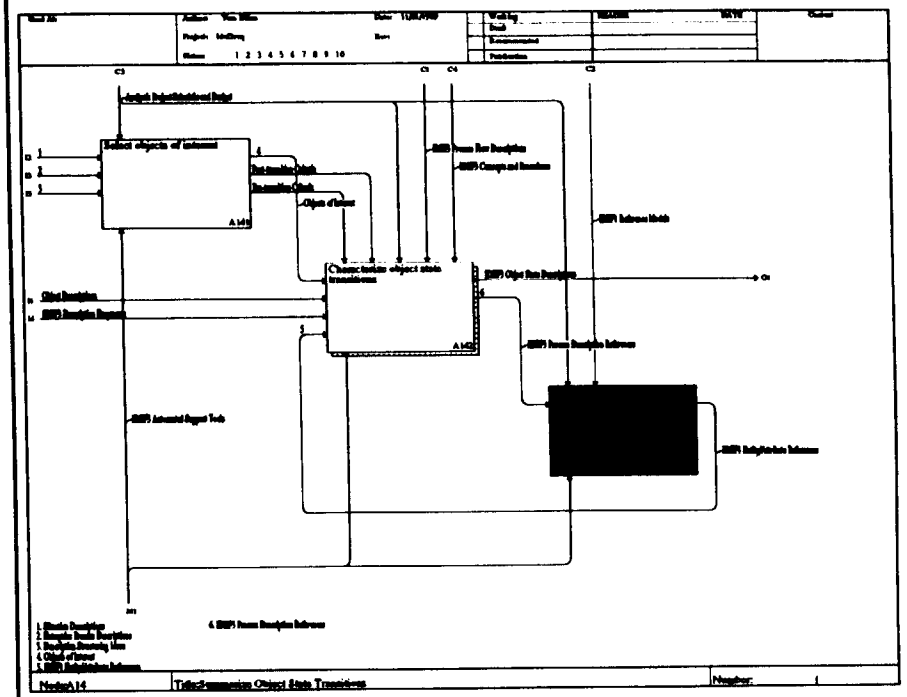
This activity identifies those UOBs that are activated when this object state transition terminates.



ORIGINAL PAGE IS
OF POOR QUALITY

Cross Reference to IDEF1 models

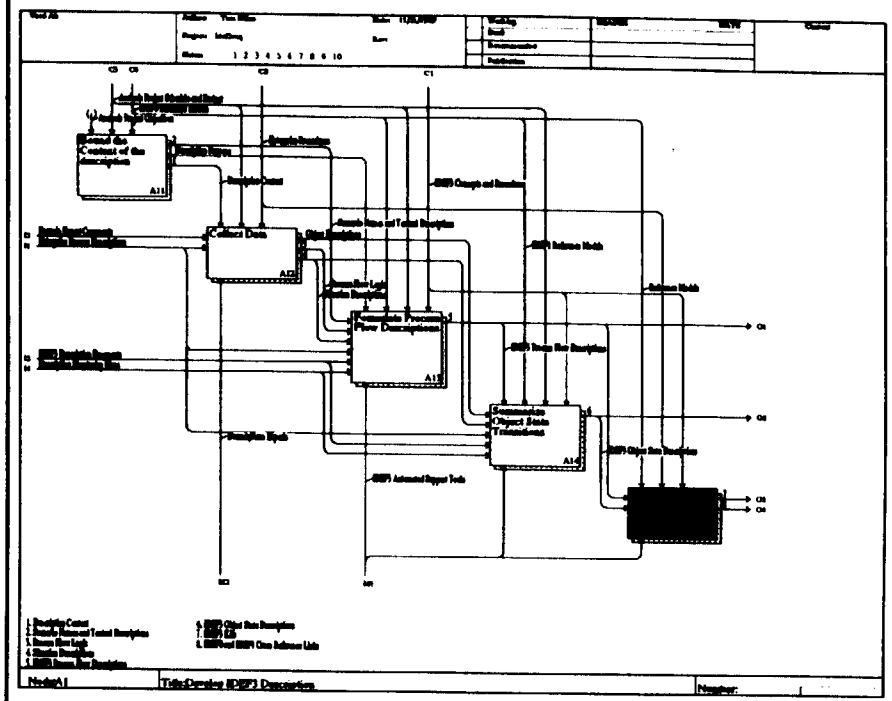
Objects that are considered in the object state transition descriptions often are defined in IDEF1 models. This activity links the objects in the transitions with their defining IDEF1 models.



ORIGINAL PAGE IS
OF POOR QUALITY

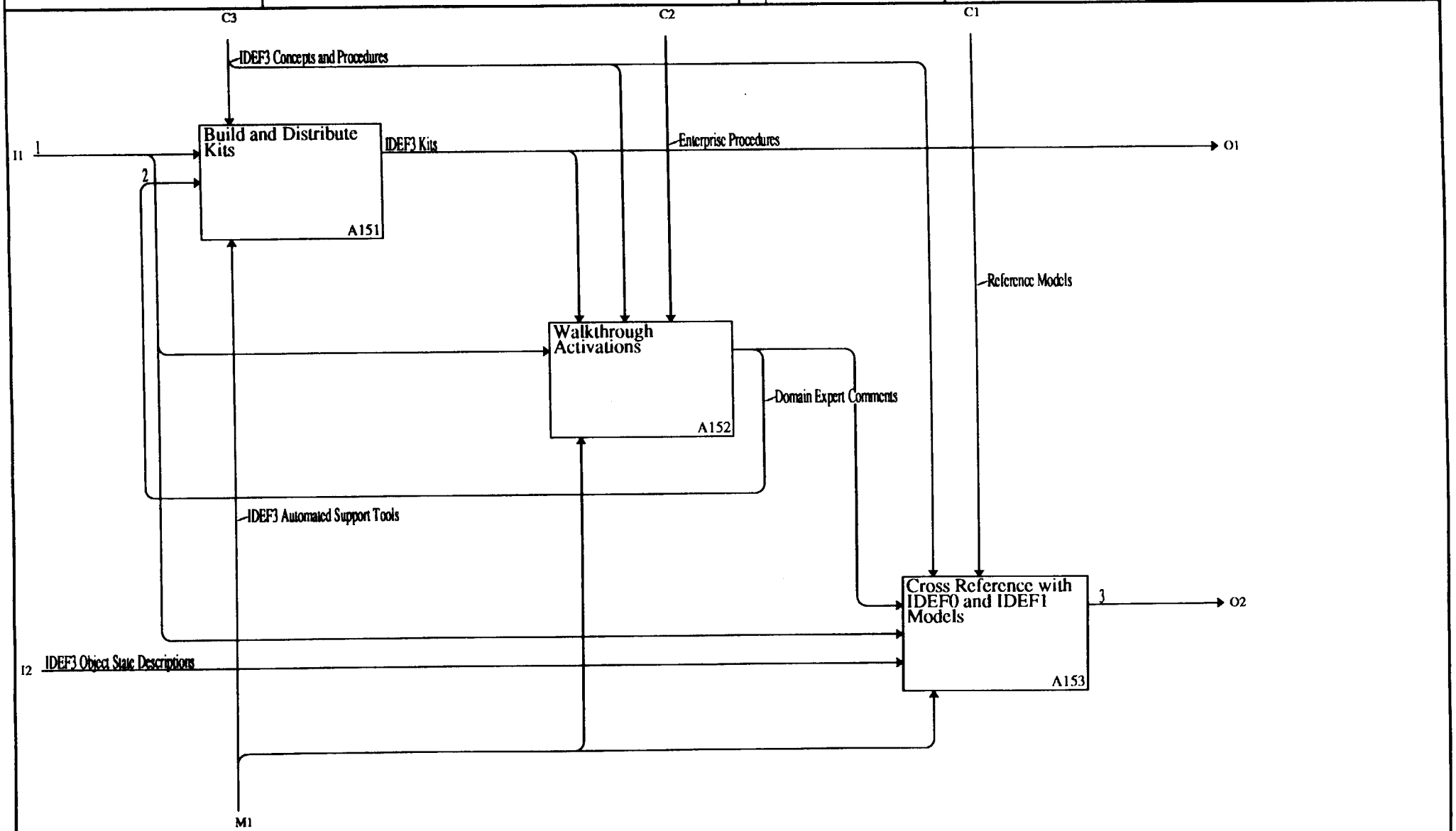
Validate IDEF3 Description

Within this activity, operations that transform the process description into a valid model occur.



ORIGINAL PAGE IS
OF POOR QUALITY

Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef3req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

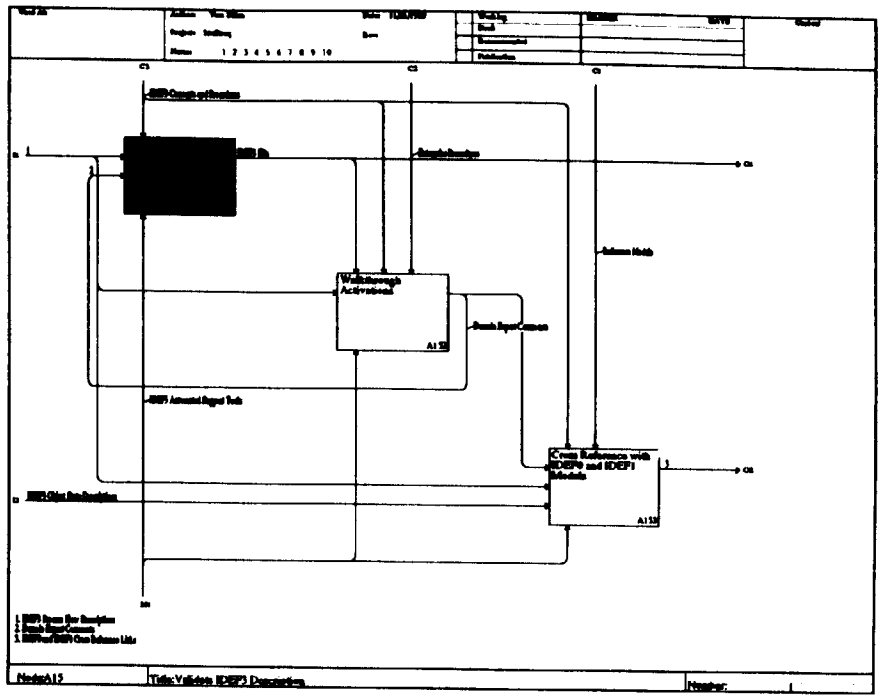


1. IDEF3 Process Flow Descriptions
2. Domain Expert Comments
3. IDEF0 and IDEF1 Cross Reference Links

Node:A15	Title:Validate IDEF3 Description	Number: 43

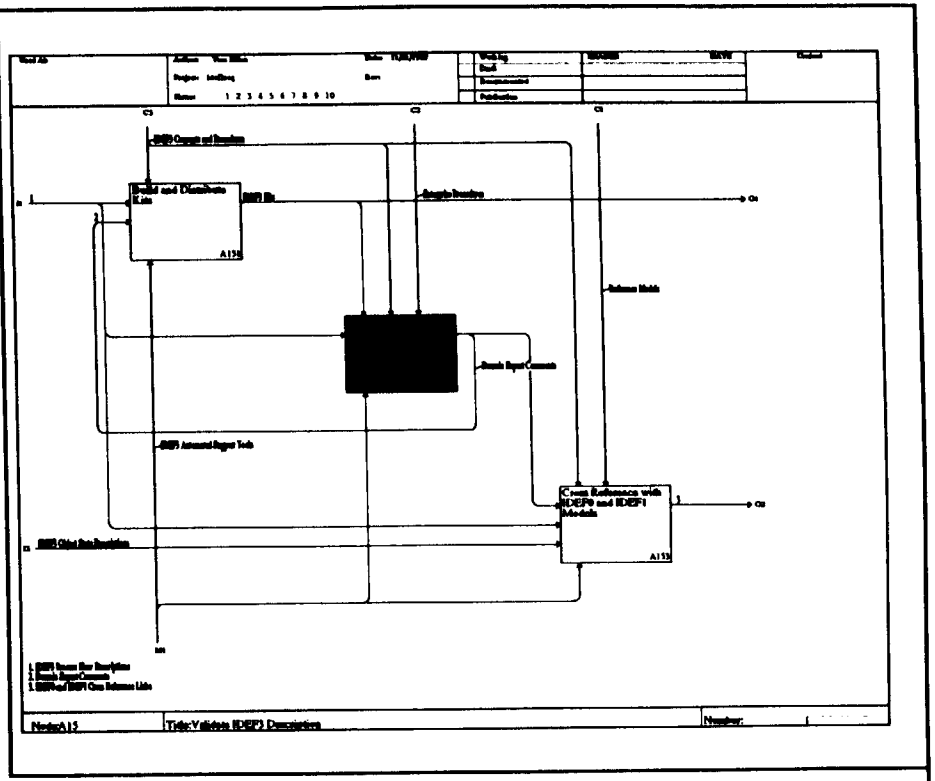
Build and Distribute Kits

This activity makes the components of the IDEF3 description available for use in other description models.



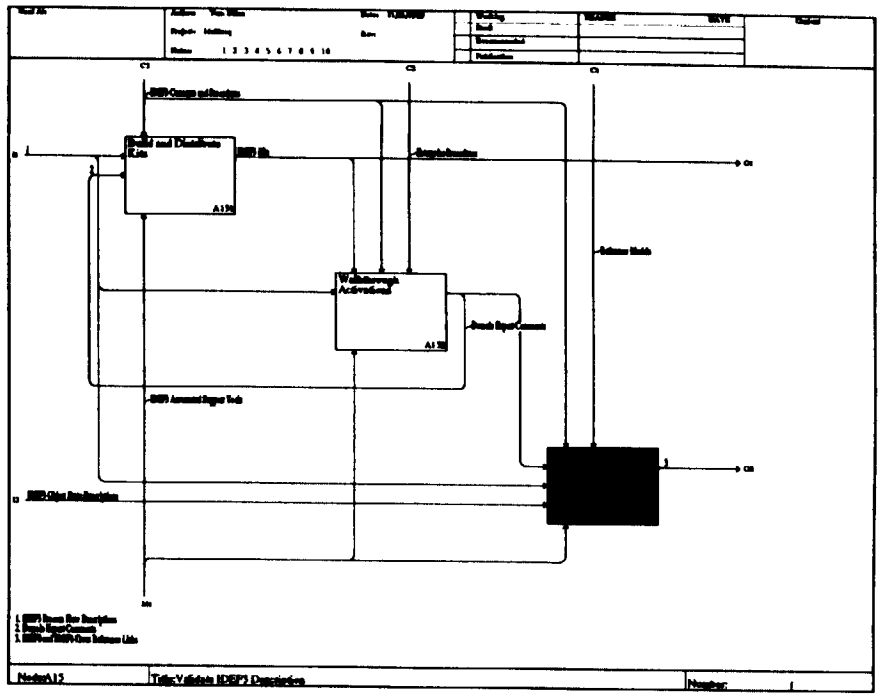
Walkthrough Activations

An activation is essentially a simple simulation of the process. It is performed to ensure that the process executes as is intended. This activity provides feedback for the other activities at this level.



Cross Reference with IDEF0 and IDEF1 Models

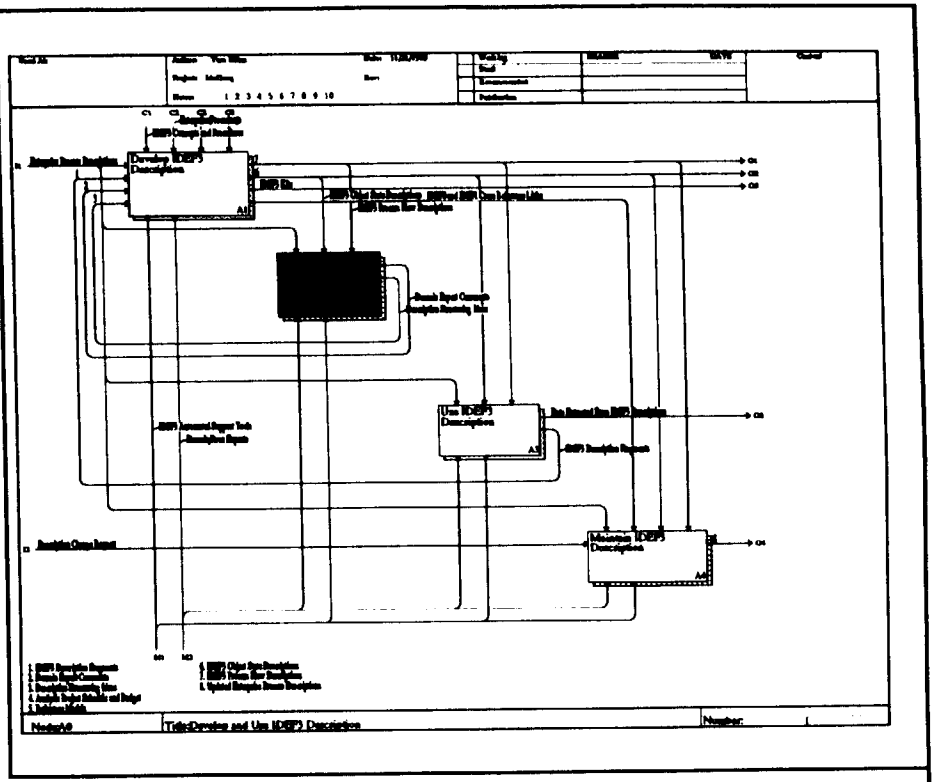
References with IDEF0 and IDEF1 models are analyzed to ensure that the models do exist and are in agreement with the information in the IDEF3 process description.



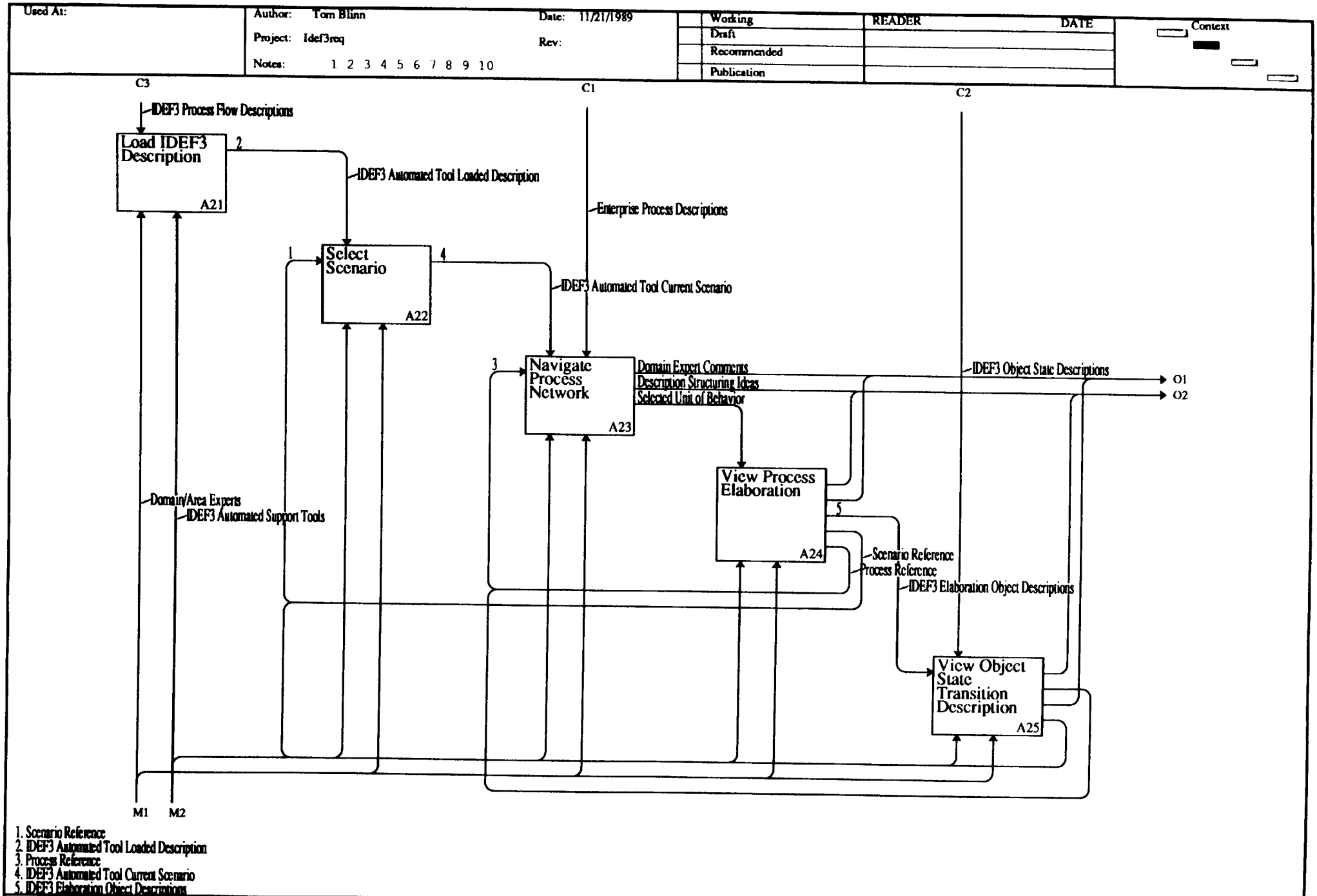
ORIGINAL PAGE IS
OF POOR QUALITY

Browse IDEF3 Description

This function is meant to convey all of the typical activities that the average reader of an IDEF3 description might make. These activities are distiguished from the use of the IDEF3 description data for other than understanding the process description (such as performing a simulation model or generation of executable code.) Domain experts performing description validation or use would be one primary class of users of this function. IDEF3 authors wishing to copy or reference portions of one description to develop another would also be primary users of this function. We would also expect that IDEF0 and IDEF1 model authors would make heavy use of the browse, copy and edit functions in this area.

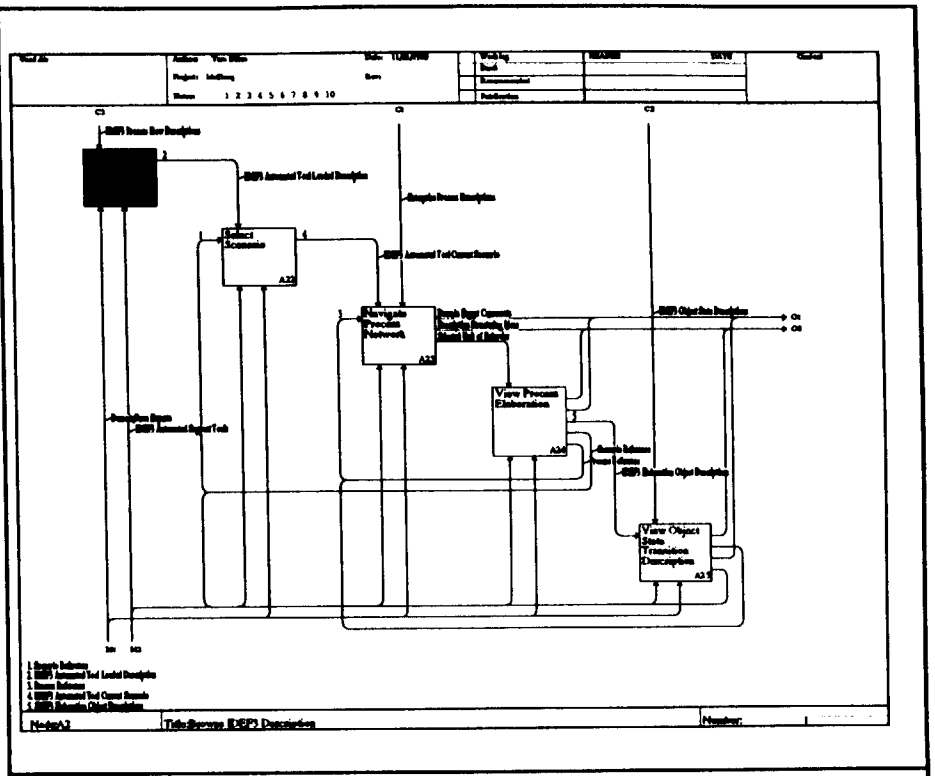


OF POOR QUALITY


Used At:
Author: Tom Blinn
Date: 11/21/1989
Project: Idef3req
Rev:
Notes: 1 2 3 4 5 6 7 8 9 10
Working
Draft
Recommended
Publication
READER
DATE
Context
C3
C1
C2
IDEF3 Process Flow Descriptions
Load IDEF3 Description
A21
2
IDEF3 Automated Tool Loaded Description
Enterprise Process Descriptions
1
Select Scenario
A22
4
IDEF3 Automated Tool Current Scenario
3
Navigate Process Network
A23
Domain Expert Comments
Description Structuring Ideas
Selected Unit of Behavior
IDEF3 Object Static Descriptions
O1
O2
View Process Elaboration
A24
5
Scenario Reference
Process Reference
IDEF3 Elaboration Object Descriptions
View Object State Transition Description
A25
Domain/Area Experts
IDEF3 Automated Support Tools
M1
M2
1. Scenario Reference
2. IDEF3 Automated Tool Loaded Description
3. Process Reference
4. IDEF3 Automated Tool Current Scenario
5. IDEF3 Elaboration Object Descriptions
Node:A2
Title:Browse IDEF3 Description
Number: 48

Load IDEF3 Description

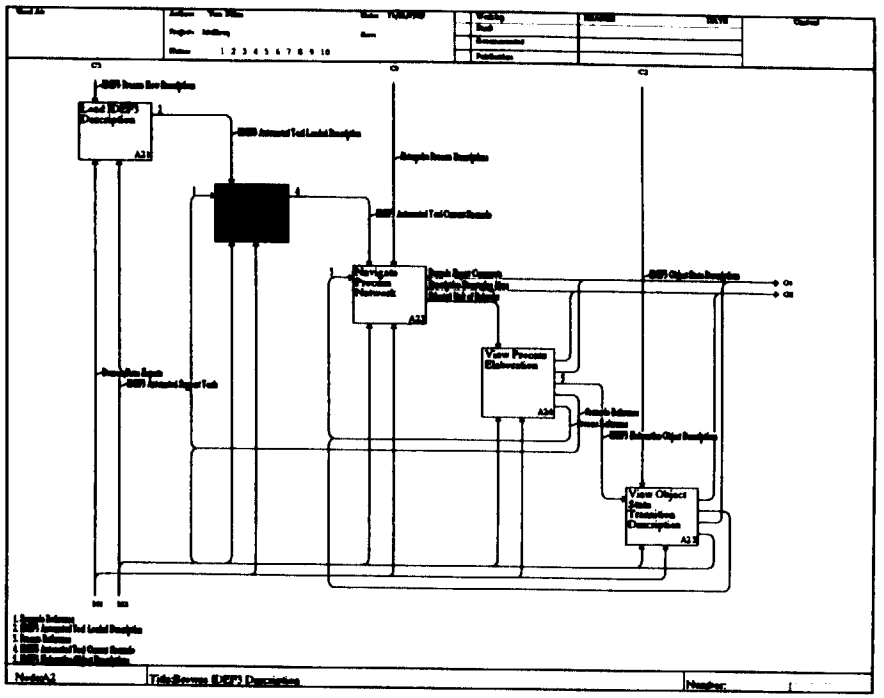
This activity load a specific IDEF3 process description into the automated tool environment.



ORIGINAL PAGE IS
OF POOR QUALITY

Select Scenario

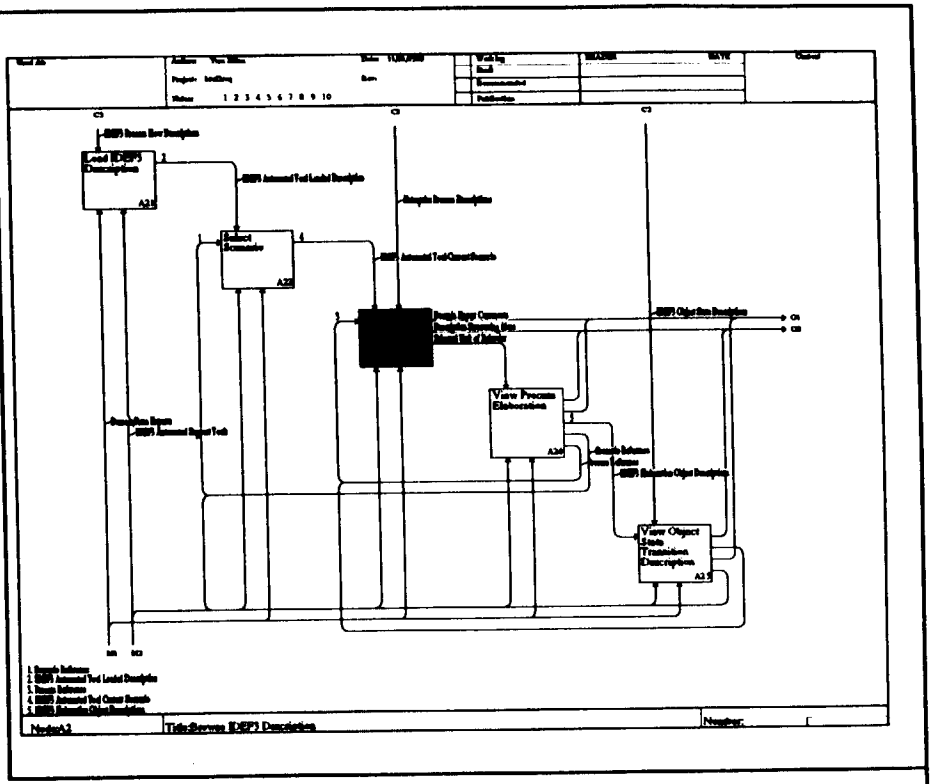
After a model is loaded, the various scenarios defined within that model are available for browsing.



ORIGINAL PAGE IS
OF POOR QUALITY

Navigate Process Network

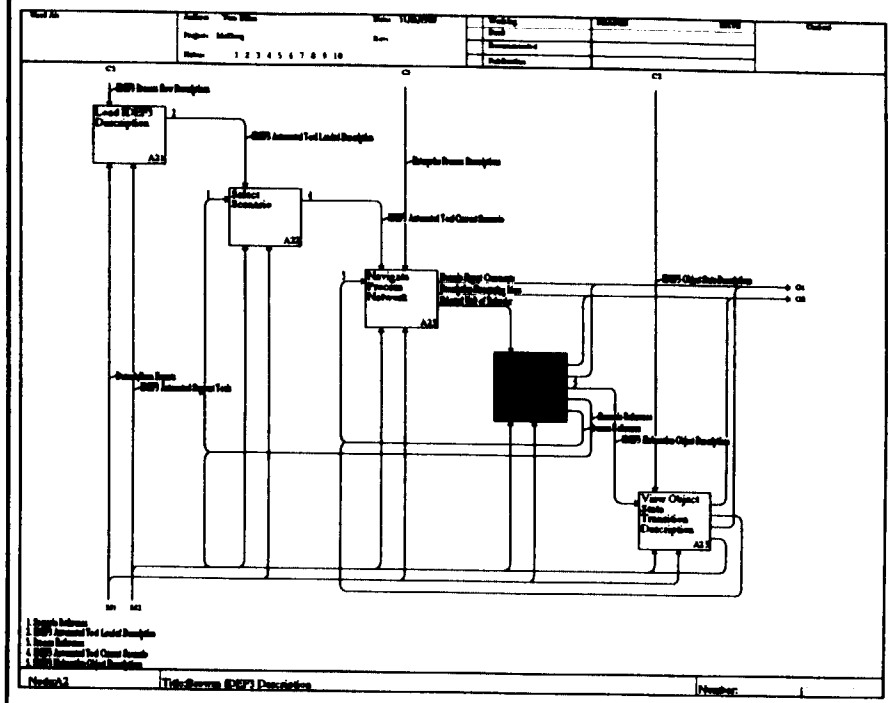
This activity simply refers to the situation of examining a process description by moving through the process flow as well as the process network by accessing the decompositions of UOBs.



BEST AVAILABLE COPY
PAGE IS OF POOR QUALITY

View Process Elaboration

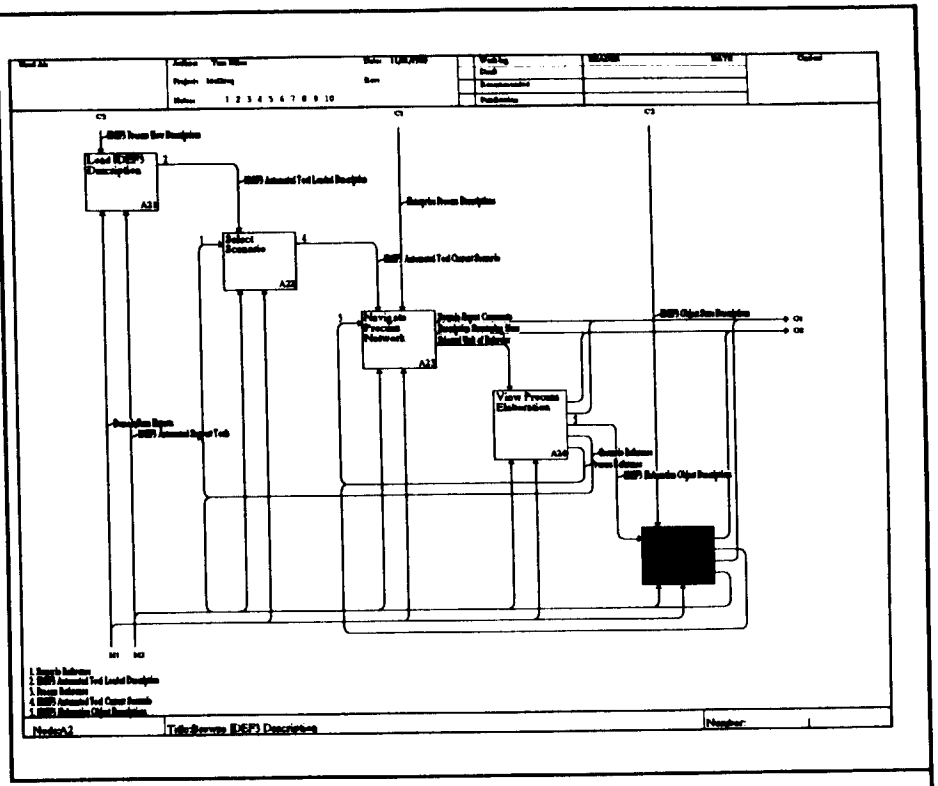
This activity indicates the ability to examine the elaborations of UOB while browsing and IDEF3 process description.



ORIGINAL PAGE IS
OF POOR QUALITY

View Object State Transition Description

Browsing also provides that ability to reference and examine the transition descriptions in the description.



ORIGINAL FOUND IN POOR QUALITY

Use IDEF3 Description

This activity represents the use of the data in an IDEF3 description for some system development decision making process. Obviously all such possible uses cannot be identified. Therefore what we have tried to represent in the decomposition of this node is the general kinds of data extraction functions which might be required from an IDEF3 description data base for such decision making processes.

Typical uses envisioned include:

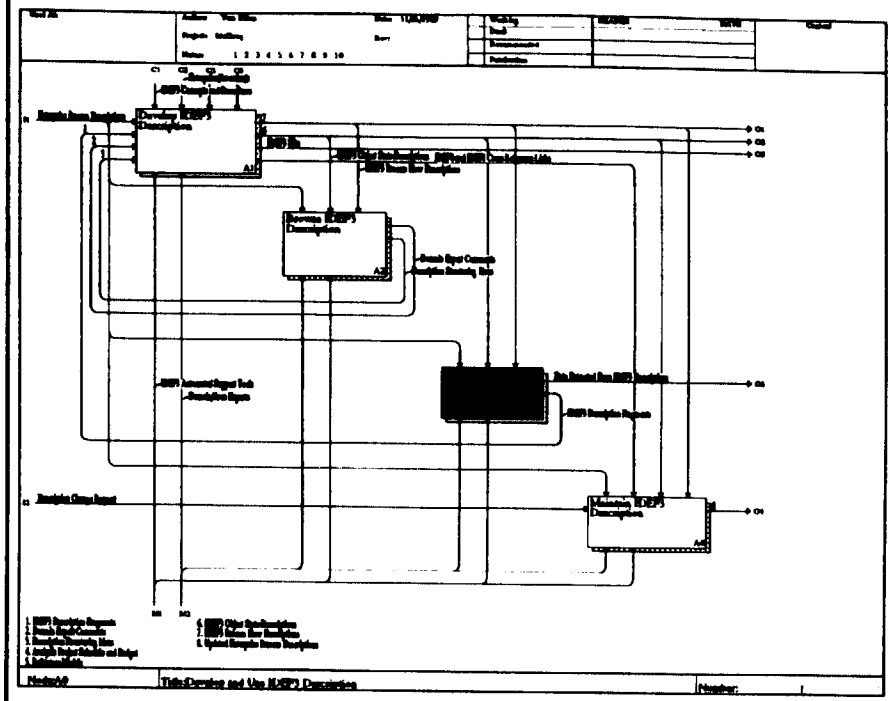
1) Organization of the results of enterprise analysis data collection.

2) Recording performance history and goals as a basis for planning and monitoring a system improvement processes.

3) External-Constraint driven design

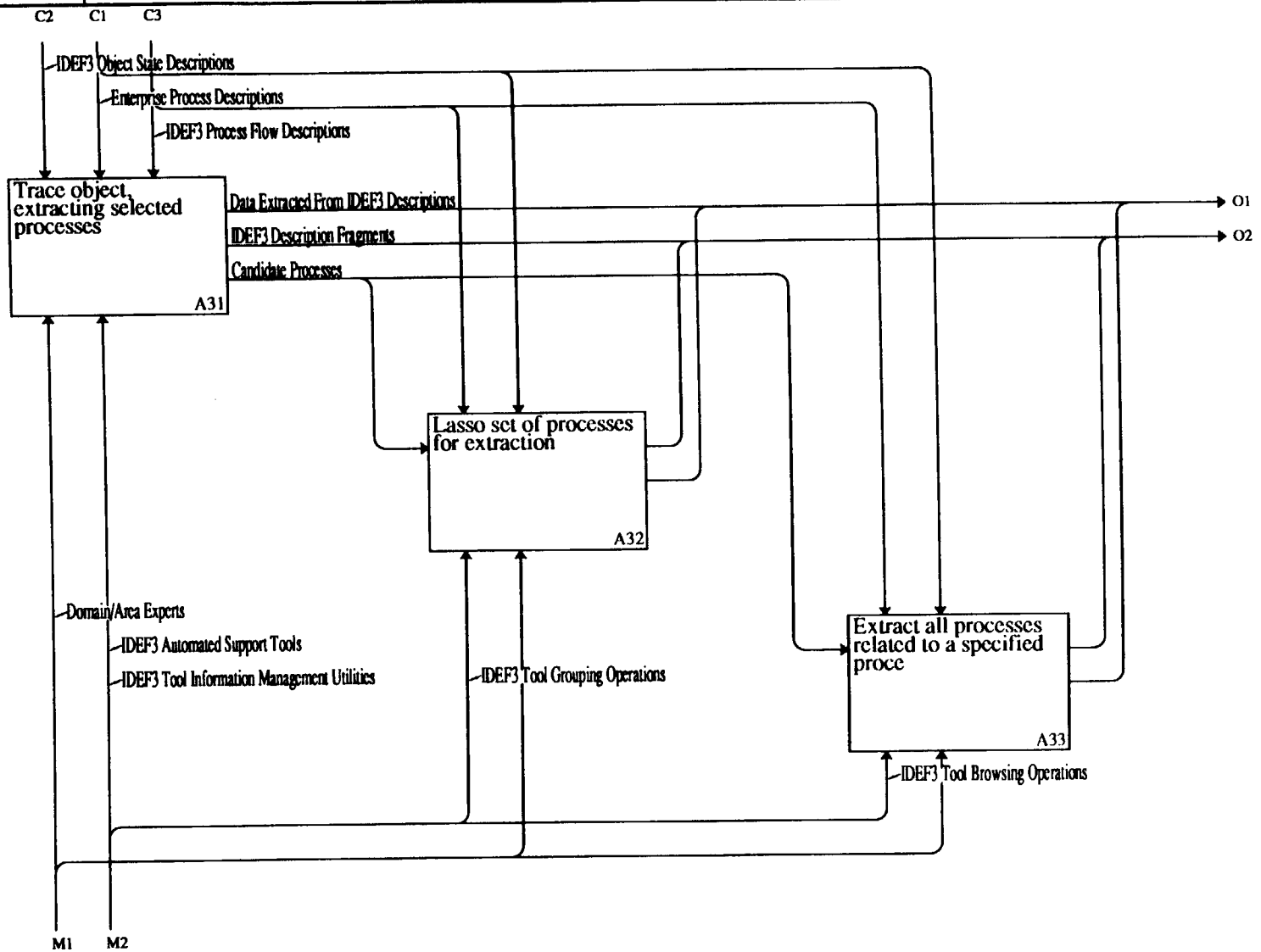
4) Material or object flow description

5) Technical or administrative data control proceedure description.



ORIGINAL PAGE IS
OF POOR QUALITY

Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef3req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

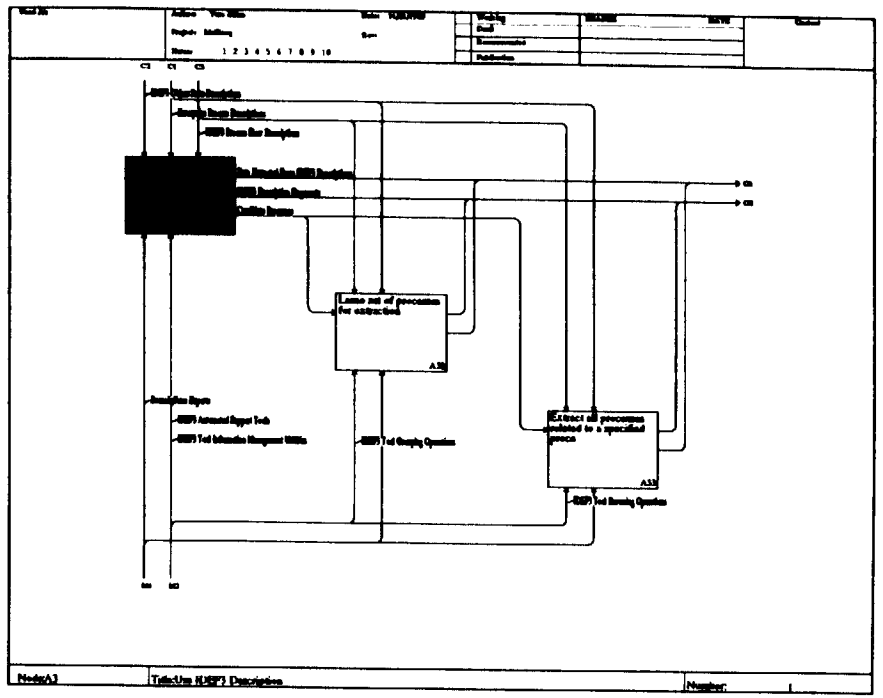


OF POOR QUALITY

Node:A3	Title:Use IDEF3 Description	Number: 55

Trace object, extracting selected processes

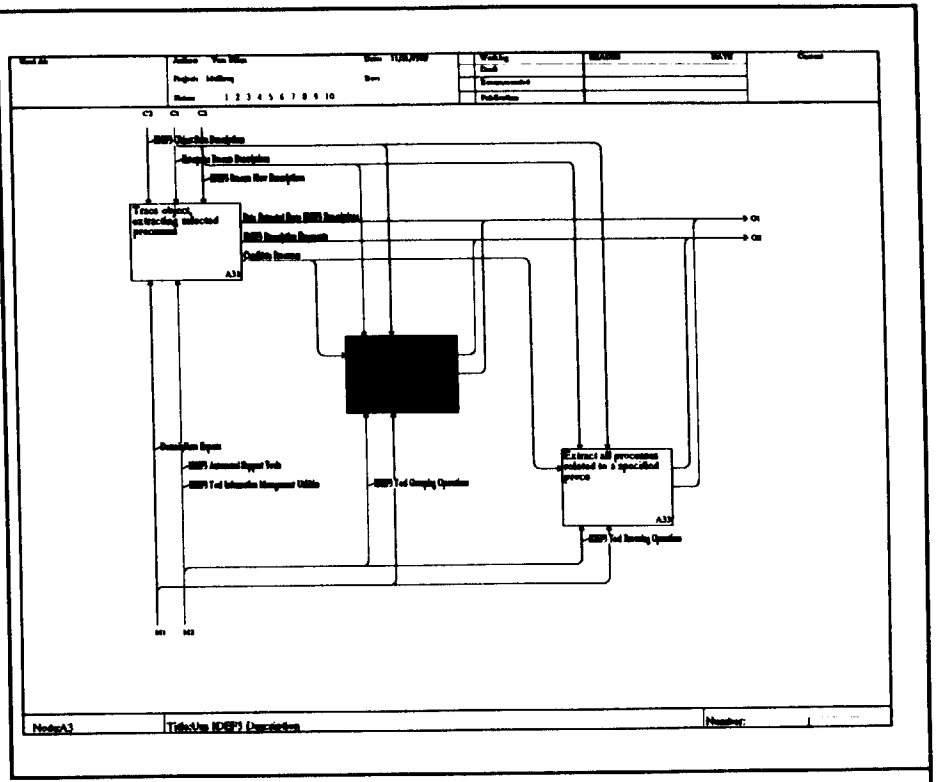
This function represents the user desiring to be able to select an object in an IDEF3 model, and extract all of the processes related to that function by some specified criteria (e.g. all processes in which the object changes state, etc.)



ORIGINAL PAGE IS
OF POOR QUALITY

Lasso set of processes for extraction

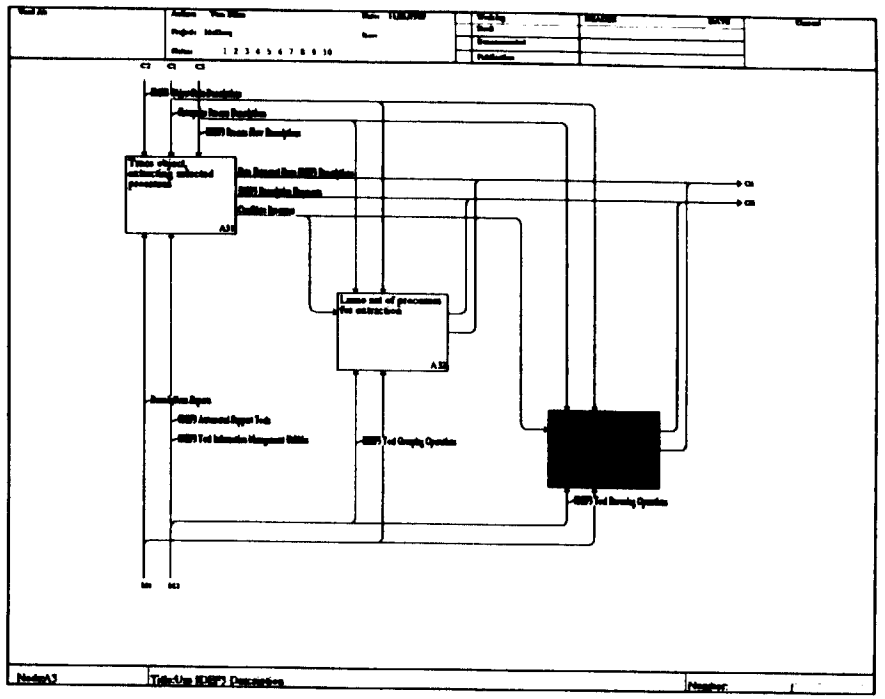
Simultaneously select a number of processes and extract them from an IDEF3 description.



ORIGINAL PAGE IS
OF POOR QUALITY

Extract all processes related to a specified proce

The function of selecting a process as the focus for a data extraction process. Based on a relation definition all of the processes related by that relation definition would be extracted from an IDEF3 description. Quite often we can expect the relation definitions to be recursive (e.g. extract all processes that the selected process stands in a precedence relationship to as well as all of the process that those extracted stand in a precedence relationship to.



ORIGINAL PAGE IS
OF POOR QUALITY

Maintain IDEF3 Description

The maintenance of an IDEF3 description includes:

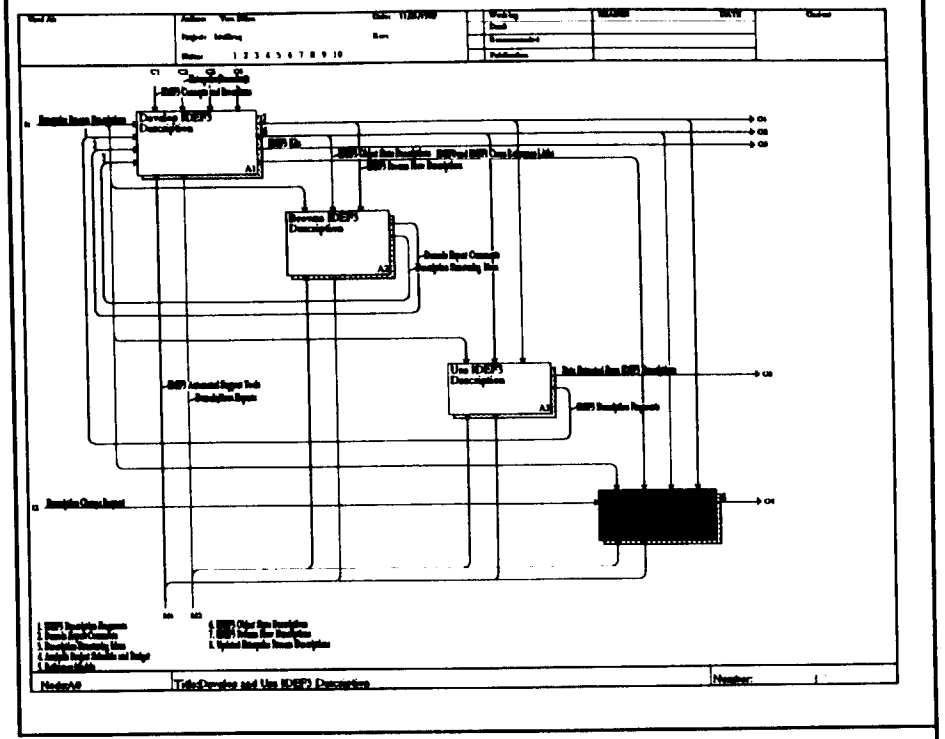
1) Changes to the individual UOB title, elaboration or IDEF0 cross reference.

2) Changing the scenario to which a UOB participates.

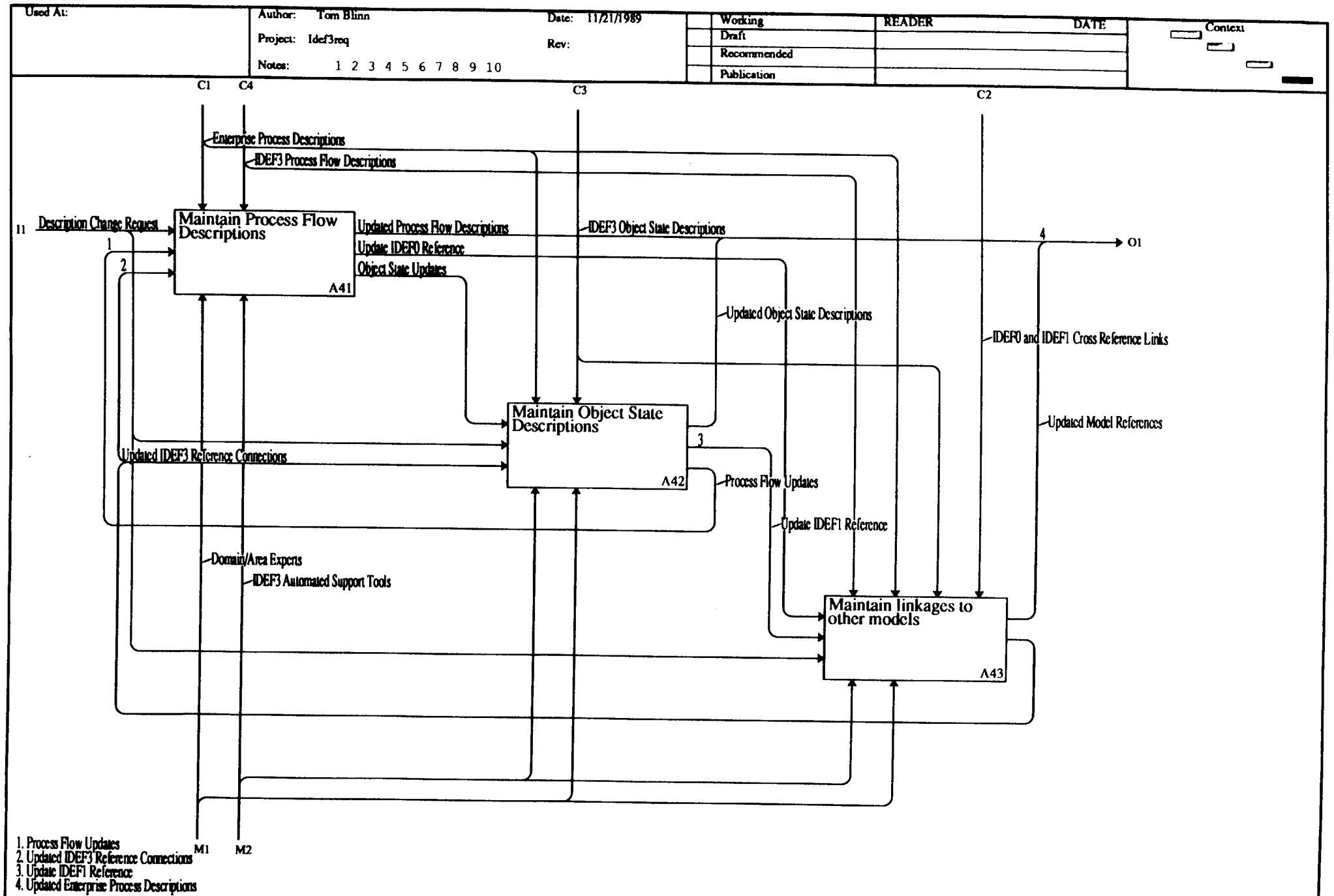
3) Changes to the set of antecedents or consequent UOBs associated with a junction.

4) Addition of new scenarios, UOBs, links, junctions, or decompositions.

5) Deletions of scenarios, UOBs, links, junctions, or decompositions.



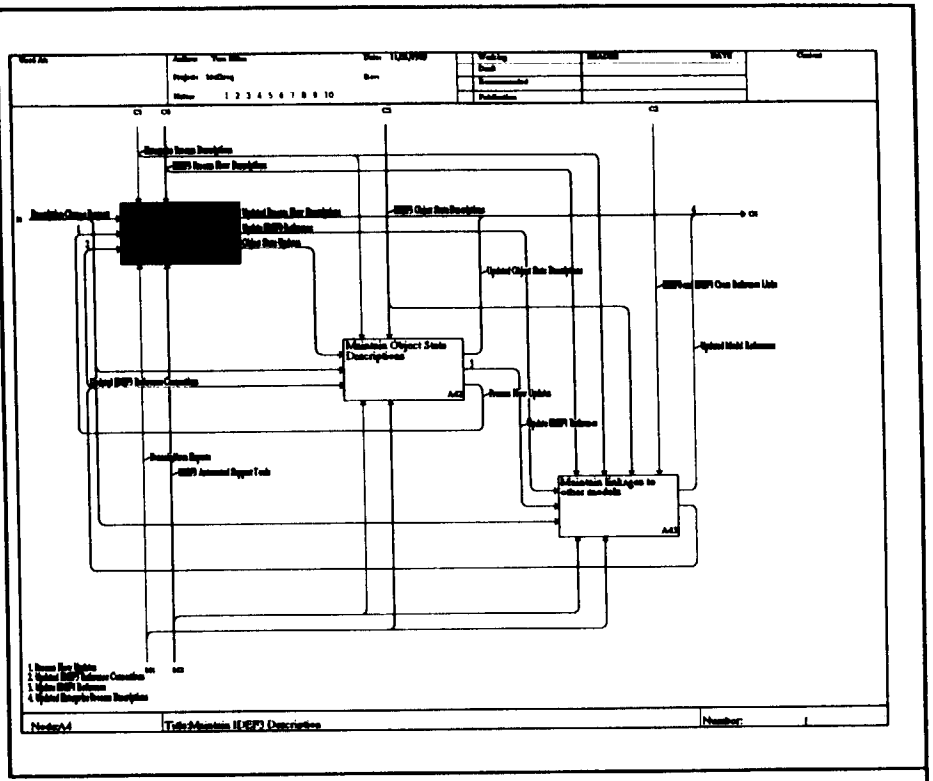
ORIGINAL PAGE IS
OF POOR QUALITY


Used At:
Author: Tom Blinn
Date: 11/21/1989
Project: Idef3req
Rev:
Notes: 1 2 3 4 5 6 7 8 9 10
Working
Draft
Recommended
Publication
READER
DATE
Context
C1
C4
C3
C2
Enterprise Process Descriptions
IDEF3 Process Flow Descriptions
I1
Description Change Request
Maintain Process Flow Descriptions
A41
Updated Process Flow Descriptions
Update IDEF0 Reference
Object State Updates
IDEF3 Object State Descriptions
4
O1
Updated Object State Descriptions
IDEF0 and IDEF1 Cross Reference Links
Maintain Object State Descriptions
A42
3
Updated Model References
Updated IDEF3 Reference Connections
Process Flow Updates
Update IDEF1 Reference
Domain/Area Experts
IDEF3 Automated Support Tools
Maintain linkages to other models
A43
M1
M2
1. Process Flow Updates
2. Updated IDEF3 Reference Connections
3. Update IDEF1 Reference
4. Updated Enterprise Process Descriptions
Node:A4
Title:Maintain IDEF3 Description
Number: 60

Maintain Process Flow Descriptions

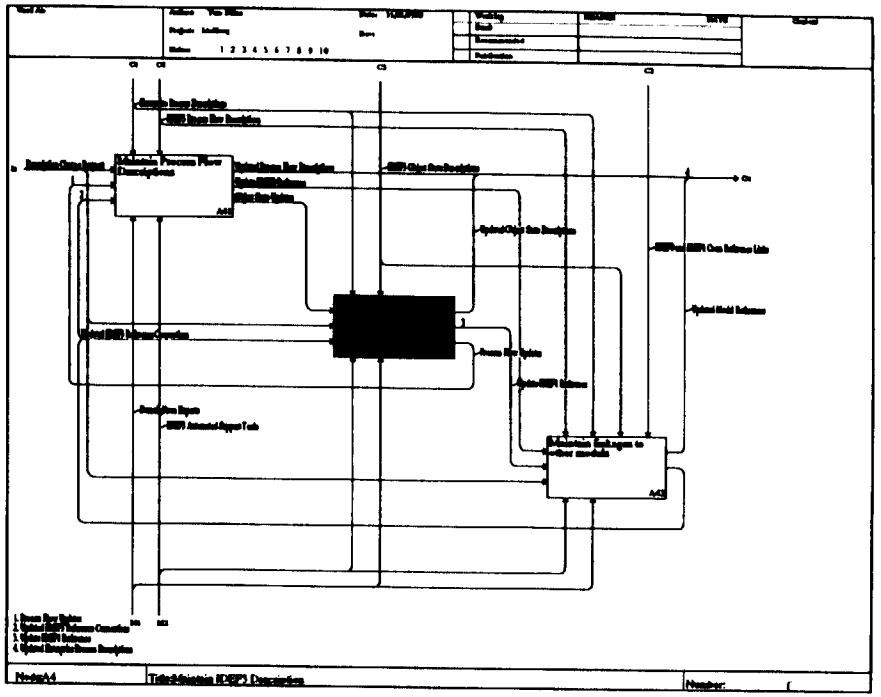
This activity encompasses all operations that are performed on an existing process description. It includes, but is not limited to:

1)Adding/editing/deleting UOBs
2)Adding/editing/deleting Junctions
3)Adding/editing/deleting Links
4)Adding/editing/deleting Decompositions
5)Adding/editing/deleting Elaborations



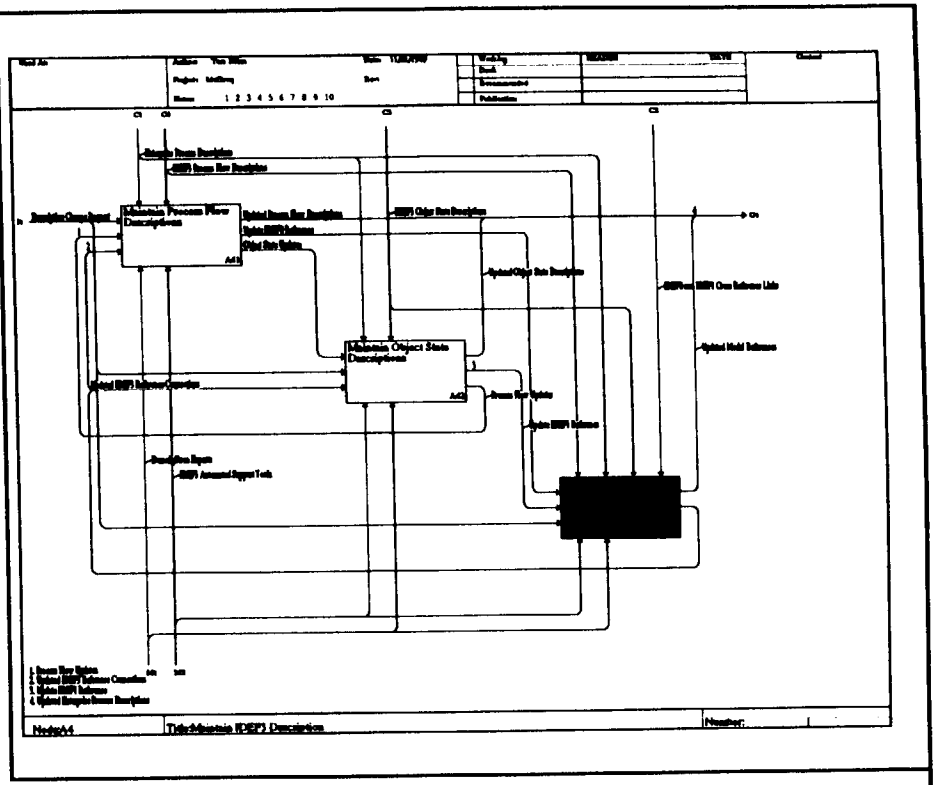
Maintain Object State Descriptions

This activity encompasses those operation involved with modifying an object state description.



Maintain linkages to other models

Linkages to other IDEF0, IDEF1, or IDEF3 models will need to be maintained as the description evolves. This maintenance requires the tracking of the linkages between the models and the issuance of notifications when changes affect models extracted from or made reference to.



GLOSSARY

Analysis Project Objectives

This concept relates the goals of the analysis of the enterprise process.

Analysis Project Schedule and Budget

This concept represents the time and monetary constraints that exist for the analysis project.

Candidate Processes

This concept represents a list of UOBs that have been highlighted within a particular process description for meeting some user-defined criteria.

Causality Relations

Data Extracted From IDEF3 Descriptions

This concept refers to the information that can be derived from any part of an IDEF3 process description.

Description Change Request

This represents a formal request to make a modification to a validated IDEF3 process description.

Description Context

This represents the scope that the process description must be developed in. It usually determines the detail to which the description is developed.

Description Purpose

The goals of the description generation are represented by this concept. It maintains what is to be accomplished with the completed process description.

Description Structuring Ideas

This concept represents some preliminary ideas on how the process flow description can be structured.

Domain Expert Comments

These comments play a vital role in the development process. The comments provide the majority of input to the description development.

Domain/Area Experts
These are the people that are most familiar with the process being modeled.

Enterprise Procedures
These procedures are the processes that an organization performs to achieve certain tasks. It is these procedures that are being modeled in the process description.

Enterprise Process Descriptions
These descriptions are the documents that outline the enterprise procedures.

IDEF0 and IDEF1 Cross Reference Links
These links provide references from IDEF3 process description objects to information maintained in other IDEF0 and IDEF1 models.

IDEF0 Reference Models
These are IDEF0 models that contain information that is pertinent to the process description being developed.

IDEF1 Entity/Attribute References
These are entity descriptions as maintained in an IDEF1 reference model.

IDEF1 Reference Models
These models have been referenced as containing information related to the process description being developed.

IDEF3 Automated Support Tools
This concept refers to the software systems used to automate the IDEF3 process description methodology.

IDEF3 Automated Tool Current Scenario

This refers to the scenario in a process model that is currently loaded in memory, displayed by the tool, and is being examined by the user.

IDEF3 Automated Tool Loaded Description

This concept refers to a process flow description that has been loaded from the database into the tool environment. It is assumed that the description can contain several IDEF3 Scenarios.

IDEF3 Concepts and Procedures

This refers to the syntax, semantics, and requirements of the IDEF3 methodology.

IDEF3 Decompositions

A decomposition in IDEF3 is very similar to a decomposition in IDEF0. The decomposition presents a more detailed description of a UOB, in terms of other UOBs, junctions, and links.

IDEF3 Description Fragments

This refers to partial descriptions that have been copied for use in other descriptions or scenarios.

IDEF3 Elaboration Object Descriptions

These descriptions provide initial information on the elaboration objects and provide references to relevant object state transition diagrams.

IDEF3 Elaborations

An elaboration is a description of a UOB in terms of the objects that exist during the execution of that UOB.

IDEF3 Junctions

Junctions provide the synchronization and logic for branches in a process description.

IDEF3 Kits

Kits are submodels that can be created for use in other models. If a portion of one model is identical to a process in another model, that model portion can be Kit saved

and then reloaded into the other model.

IDEF3 Object Flow Relationship

An IDEF3 Object Flow Relationship simply indicates that an object that exists during one UOB will continue to existing in the upcoming UOB.

IDEF3 Object State Descriptions

An object state description describes the object characteristics that define the particular constraints. Simply, it defines the conditions that must be met before the state is acquired.

IDEF3 Object States

These are the various object states that exist within a process description.

IDEF3 Objective View

IDEF3 Post-transition Requirements

These requirements are the IDEF3 definitions of the post-transtion criteria for the object state.

IDEF3 Pre-transition Requirements

These requirements are the IDEF3 definitions of the pre-transition criteria of the object state.

IDEF3 Precedence Relations

These relations indicate the sequencing of UOBs within a process description.

IDEF3 Process Description References

An IDEF3 Reference Connection allows a description to refer to some piece of information that does not appear in the current desription. The reference could be to another process desription, UOB, scenario, IDEF0 or IDEF1 model element, or many others.

IDEF3 Process Flow Descriptions

A process flow description is a description of how things work, in terms of processes and objects that exist within those processes.

IDEF3 Scenario

A Scenario presents a certain perspective of a process description. A description of a process can consist of many different scenarios.

IDEF3 Tool Browsing Operations

The Browsing Operations provide the ability to move around the various scenarios, process descriptions, and object state transition descriptions.

IDEF3 Tool Grouping Operations

Tool Grouping supports the selection of multiple objects in a process description so that operations may be performed on the entire group.

IDEF3 Tool Information Management Utilities

The Information Management provides the functions necessary for the cross references required in a process description. For example, a UOB can be used in many different places and these utilities keep track of all these references.

IDEF3 Units of Behavior

The Unit of Behavior (UOB) is the base unit of a process description. It has a verb based label that indicates the nature of process being represented.

IDEF3 View

An IDEF3 View is a general UOB decomposition.

Object Descriptions

These are textual descriptions that provide information on the objects that exist within a process.

Object State Characteristics

These are textual descriptions of the characteristics that define an object state.

Object State Updates

These refer to changes that must be made to object state descriptions during the maintenance of a process description.

Objects of Interest

This is a list of objects that seem to play important roles in the processes being described in the model.

Post-transition Criteria

These criteria are descriptions of the general requirements that must be satisfied before a state transition can complete.

Pre-transition Criteria

These criteria describe the constraints on an object state that must be satisfied before a state transition can occur.

Process Flow Logic

This refers to the semantics of the links and junctions that relate the various UOBs in a process description.

Process Flow Updates

This refers to the changes that must be made to a process flow during the maintenance of a process description.

Process Reference

This concept refers to the situation where an elaboration or object state transition diagram refers to another process that the expert may be interested in examining.

Reference Models

These are models (IDEF0 and IDEF1) that are references in a process description.

Scenario Names and Textual Descriptions

These are textual descriptions of the enterprise processes.

Scenario Reference

This concept deals with the situation where an elaboration or object state transition diagram refers to another scenario that the expert may be interested in examining.

Selected Unit of Behavior

This references a specific UOB that is being examined.

Situation Descriptions

A situation description is a textual description of the current state of the process. This includes information on the objects participating in the process.

Update IDEF0 Reference

When a UOB is updated or changed, the IDEF0 activity changes as well. This requires that the reference to the IDEF0 model be updated.

Update IDEF1 Reference

When an Object State is updated, the IDEF1 reference sometimes changes as well. This requires that the IDEF1 model reference be updated as well.

Updated Enterprise Process Descriptions

This represents a validated process description that has undergone a modification.

Updated IDEF3 Reference Connections

When a reference changes, the Reference Connection must be updated as well.

Updated Model References

This refers to the model references that must be updated when references to objects maintained in those models have changed.

Updated Object State Descriptions

This refers to the new descriptions that result from modification to existing object state descriptions.

Updated Process Flow Descriptions

These are the new descriptions that result from modification to validated process descriptions.

PRECEDING PAGE BLANK NOT FILMED

Appendix B: IDEF1 Model of IDEF3

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

I.D. No.	Source Material Name	Received From
1	IDEF3 Technical Report	KBS Lab
2	IDEF3 Formalization Rpt.	KBS Lab
3	IISyCL Language Report	KBS Lab
4	IDEF0 Reference Manual	U.S. Air Force
5	IDEF1 Reference Manual	U.S. Air Force
6	IDSE Metamodel Report	KBS Lab
7	IDEF3 Automation ReqtsDoc	Tom Blinn

Node:	Title: Source Material Log	Number:

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

I.D. No.	Source Data Name	Cross Reference Source Material
1	Scenario	7, 2, 1
2	Unit of behavior	7, 2, 1
3	Situation description	7, 2, 1
4	View Decomposition	4, 7, 2, 1
5	Elaboration	7, 2, 1
6	Process Flow Description	7, 2, 1
7	Object Reference	7, 2, 1
8	Fact Declaration	7, 2, 1, 3
9	Constraint Reference	7, 2, 1, 3
10	Scenario Label	7, 2, 1
11	Scenario Glossary	7, 2, 1
12	Object Label	7, 2, 1
13	Object State	7, 2, 1
14	Specific Object	7, 2, 1
15	Non-Specific Object	7, 2, 1
16	Precidence Link	7, 2, 1
17	Relational Link	7, 2, 1
18	Object Flow Link	7, 2, 1
19	Junction	7, 2, 1
20	Synchronous AND Junction	7, 2, 1
21	Asynchronous AND Junction	7, 2, 1
22	Synchronous OR Junction	7, 2, 1
23	Asynchronous OR Junction	7, 2, 1
24	XOR Junction	7, 2, 1
25	UOB Label	7, 1
26	UOB Number	7, 2, 1
27	Objective Decomposition	1
28	Process Flow Description	7, 2, 1
29	Object State Description	7, 1
30	Process Flow Diagram	7, 2, 1
31	Object State Diagram	7, 1

Node:	Title: Source Data Log	Number:

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

Entity Class I.D. No.	Entity Class Name	Source Data I.D. No.
1	Unit of Behavior	
2	Object State	
3	Scenario	
4	Decomposition	
5	Elaboration	
6	View	
7	Process Description	
8	Object State Transition Description	
9	Junction Link	
10	Link	
11	Object State Transition Link	
12	Object	
13	Constraint Set	
14	UOB occurence in Decomposition	
15	UOB occurence in Scenario	
16	OST Precondition Spec	
17	Constraint Set in OST Precondition Spec	
18	Constraint Set in Elaboration Spec	
19	Constraint Set in Link	
20	Object in Elaboration	
21	Object in View	
22	Fact Statement	
23	Object in Fact Statement	
24	Fact Statement in Elaboration Specification	
25	Process Description used in Object State Transitn	
26	OST Postcondition Spec	
27	Constraint Set in OST Postcondition Specification	
28	Fact Use in Object State Specification	

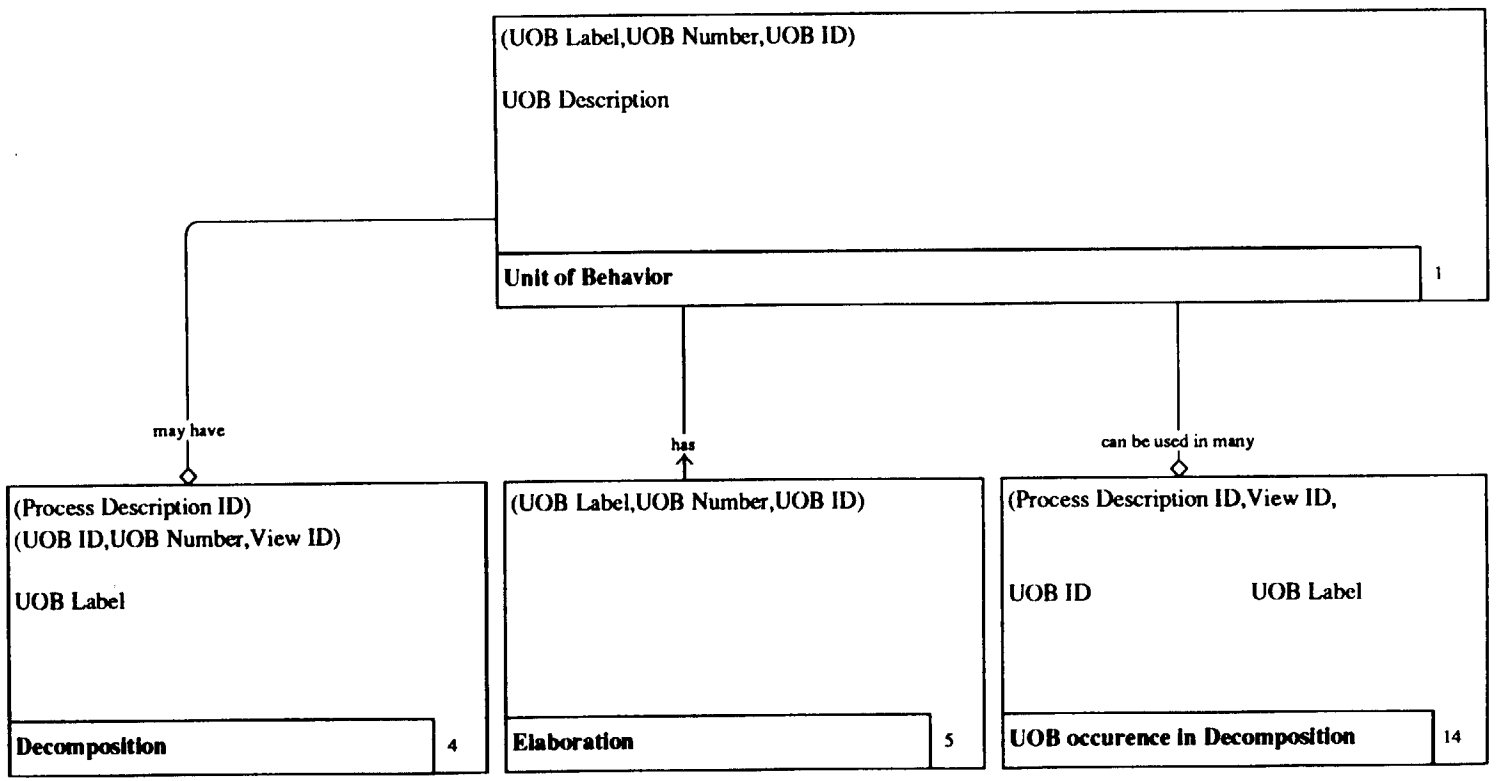
Node:	Title: Entity Class Pool	Number:

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

I.D. No.	Attribute Class Name	Source Data I.D. No.
4	View Type	
5	Junction Type	
6	Link Type	
7	Junction Spread	
8	Junction Control Type	
9	Front of link UOB	
11	UOB Label	
12	UOB Description	
13	UOB Occurence Number	
14	Back of Link UOB	
15	Reference Pointer Block Number	
16	Reference Pointer Block Type	
17	Decomposition type	
18	View Name	
19	View ID	
20	Object Name	
21	Object ID	
22	Object Description	
23	View Glossary	
24	Object State Label	
25	Object State Name	
26	Object State ID	
27	Scenario Name	
28	Scenario Glossary	
29	Scenario ID	
30	UOB ID	
31	Fact Statement ID	
32	Fact Relation	
33	Fact Polarity	
34	Process Description ID	
35	Constraint ID	
36	Constraint Statement	
37	Object Use Reference Type	

Node:	Title: Attribute Class Pool	Number:

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



Node: E1	Title: Unit of Behavior	Number: 55

PRECEDING PAGE BLANK-NOT FILMED

Entity Class Definition: The basic information managed by IDEF3 for the purpose of establishment of a description of a state of affairs or a state of change in an organization or other system environment.

Key Classes:
(UOB Label,UOB Number,UOB ID)
Owned Attribute Classes:

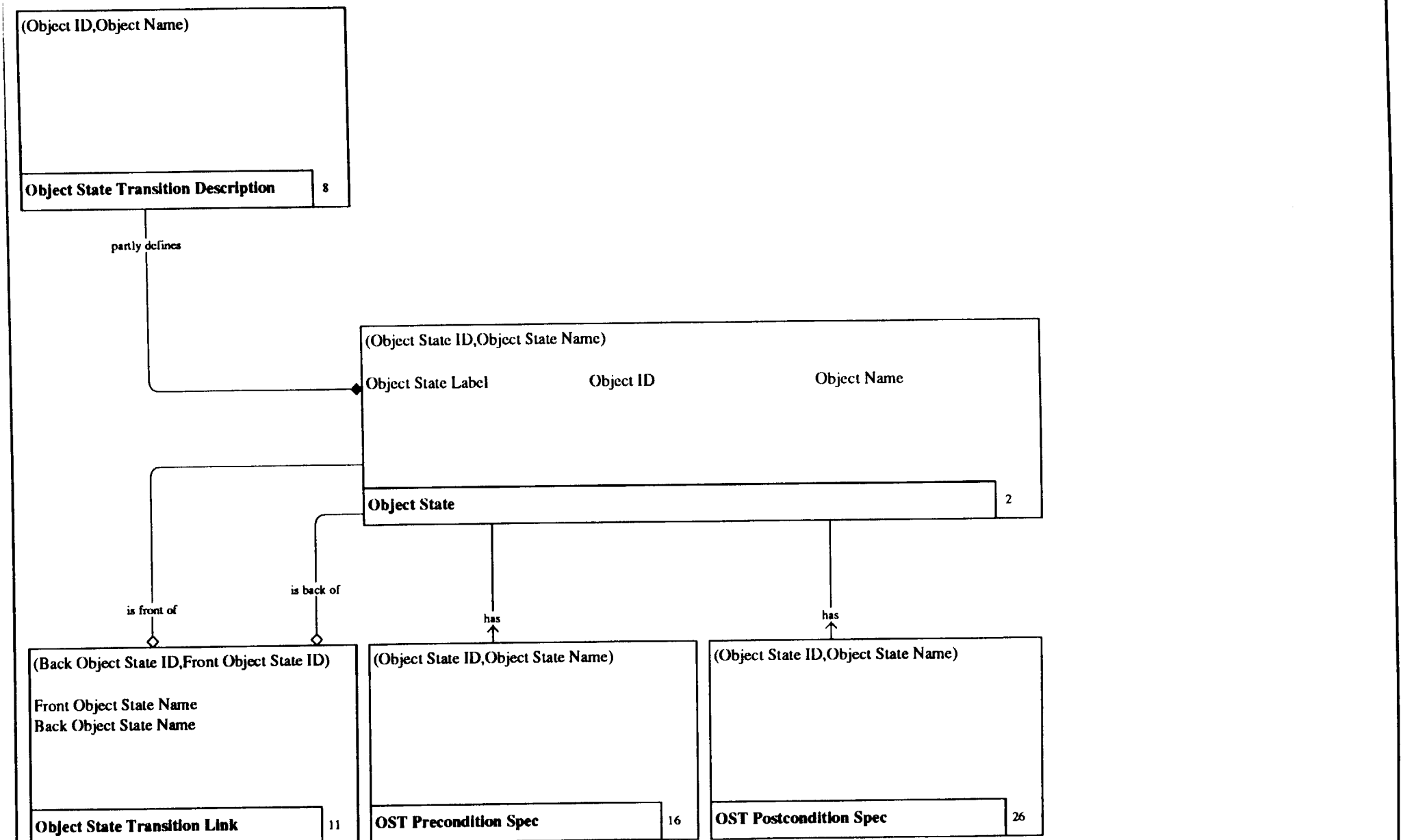
Name: UOB ID
Definition: The unique identifier of a UOB.

Name: UOB Description
Definition: This attribute records a glossary entry of a UOB.

Name: UOB Label
Definition: This attribute records the displayed label associated with a UOB.

Name: UOB Number
Definition: The unique reference number of a UOB occurence in a scenario or a decomposition.

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



Node: E2	Title: Object State	Number: 53

Entity Class Definition: The characterization of a named state of an object in an organization or other system environment.

Key Classes:
(Object State ID,Object State Name)
Owned Attribute Classes:

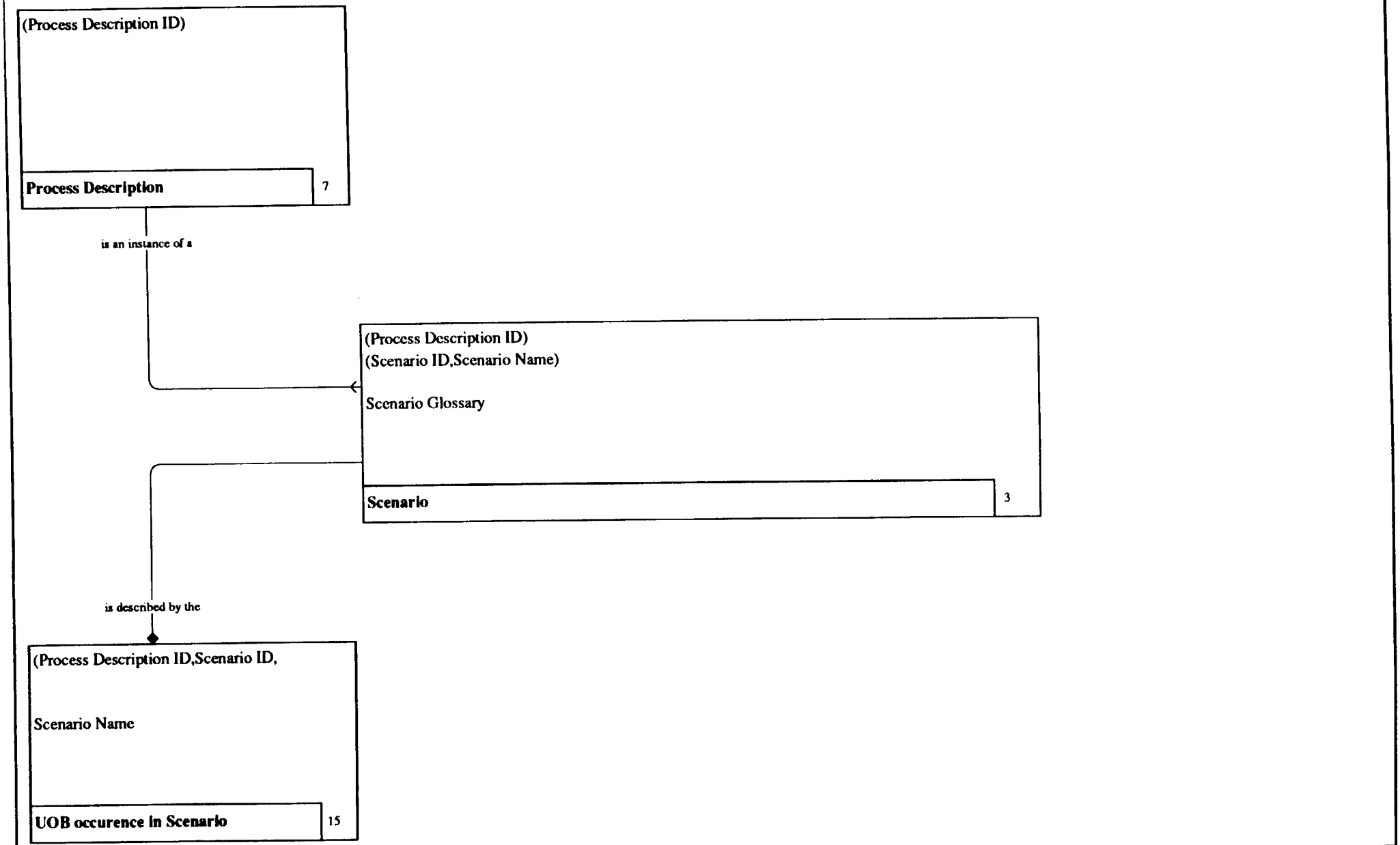
Name: Object State ID
Definition: This attribute is a unique identifier associated with each object state of an object. It is guaranteed unique across the description, and can be used to support the integration of multiple descriptions.

Name: Object State Label
Definition: This attribute records the displayed label associated with an object state in an object state diagram.

Name: Object State Name
Definition: This attribute records the unique object state name associated with each declared possible state of an object.

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Object Name	Object	12	Object State Transition Description	8	is described by
Object ID	Object	12	Object State Transition Description	8	is described by

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



Node: E3	Title: Scenario	Number: 51

Entity Class Definition: The name of a description of a reoccuring problem or typical situation that plays a major role in the organization of the activities of an organization or other system environment.

Key Classes:
(Process Description ID) (Scenario ID,Scenario Name)
Owned Attribute Classes:

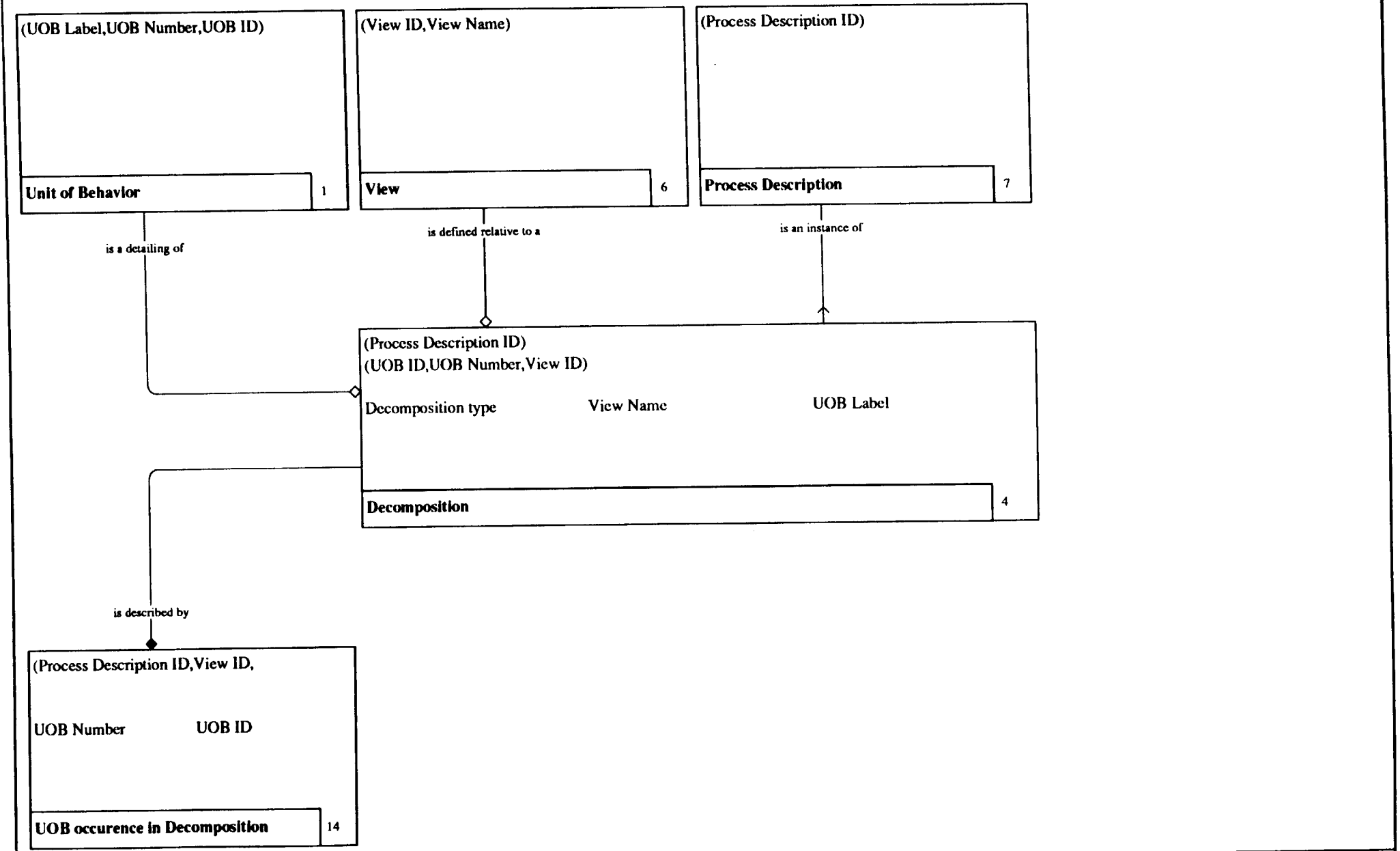
Name: Scenario Glossary
Definition: A descriptive text entry associated with a scenario name.

Name: Scenario ID
Definition: As scenarios may have the same name due to viewpoint changes the unique scenario ID allows one to unabiguously reference a scenario in an IDEF3 process description. The unique ID also supports multiple description integration.

Name: Scenario Name
Definition: The referencing name of a scenario.

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Process Description ID	Process Description	7	Process Description	7	can be instanced as a

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



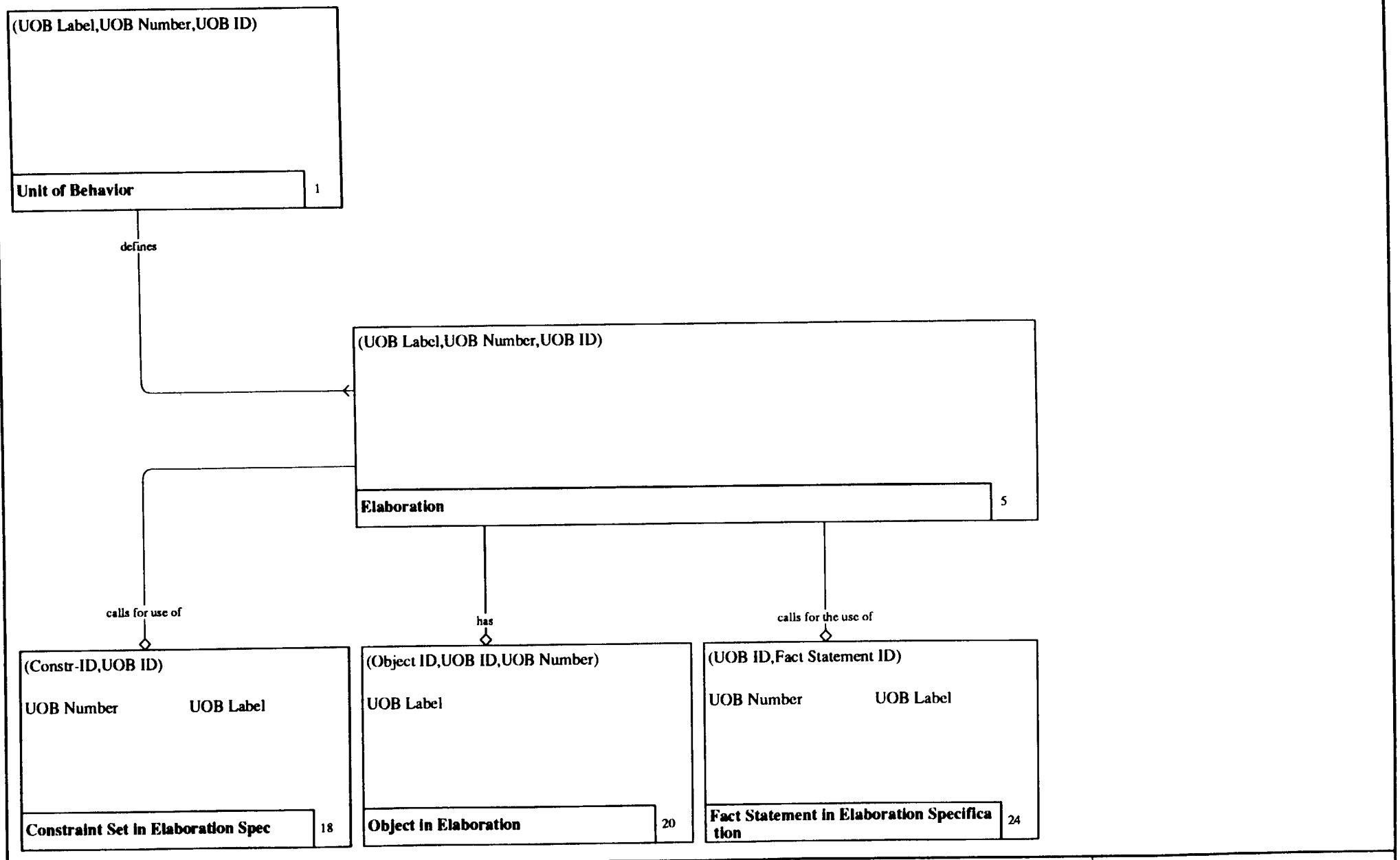
Node: E4	Title: Decomposition	Number: 49

Entity Class Definition: This entity class tracks the view diagrams linked to a UOB. In IDEF3 the process description is broken up into scenarios and decompositions. Each of these logical parts can have as many diagrams as necessary to display the set of UOBs and links that make up the content of that partition. This is unlike IDEF0 where the display diagram constrains the description.

Key Classes:
(Process Description ID) (UOB ID,UOB Number,View ID)
Owned Attribute Classes:
Name: Decomposition type
Definition: A decomposition of a UOB can be a view decomposition or an objective decomposition.

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Process Description ID	Process Description	7	Process Description	7	can be instanced as a
UOB ID	Unit of Behavior	1	Unit of Behavior	1	may have
UOB Number	Unit of Behavior	1	Unit of Behavior	1	may have
UOB Label	Unit of Behavior	1	Unit of Behavior	1	may have
View Name	View	6	View	6	describes the context of
View ID	View	6	View	6	describes the context of

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



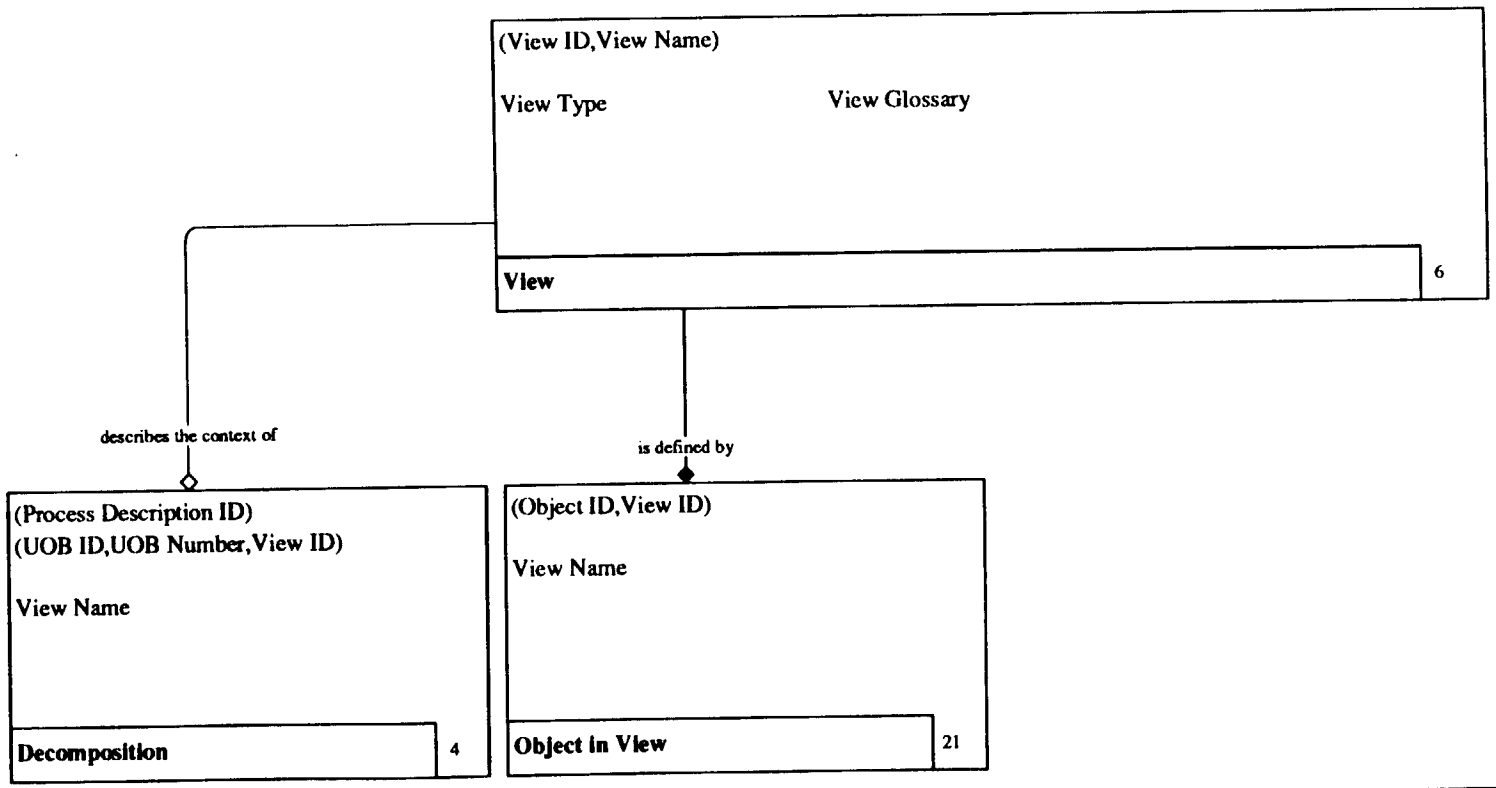
Node: E5	Title: Elaboration	Number: 47

Entity Class Definition: The component of a UOB description that specifies the objects, facts, and constraints that characterize that UOB. If a UOB has an elaboration it has one and only one elaboration, and an elaboration is used for only one UOB.

Key Classes:
(UOB Label,UOB Number,UOB ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
UOB ID	Unit of Behavior	1	Unit of Behavior	1	has
UOB Number	Unit of Behavior	1	Unit of Behavior	1	has
UOB Label	Unit of Behavior	1	Unit of Behavior	1	has

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



Node: E6	Title: View	Number: 45

Entity Class Definition: The name of a context setting structure for decomposition process descriptions associated with a unit of behavior in an IDEF3 process description. description.

Key Classes:
(View ID, View Name)
Owned Attribute Classes:

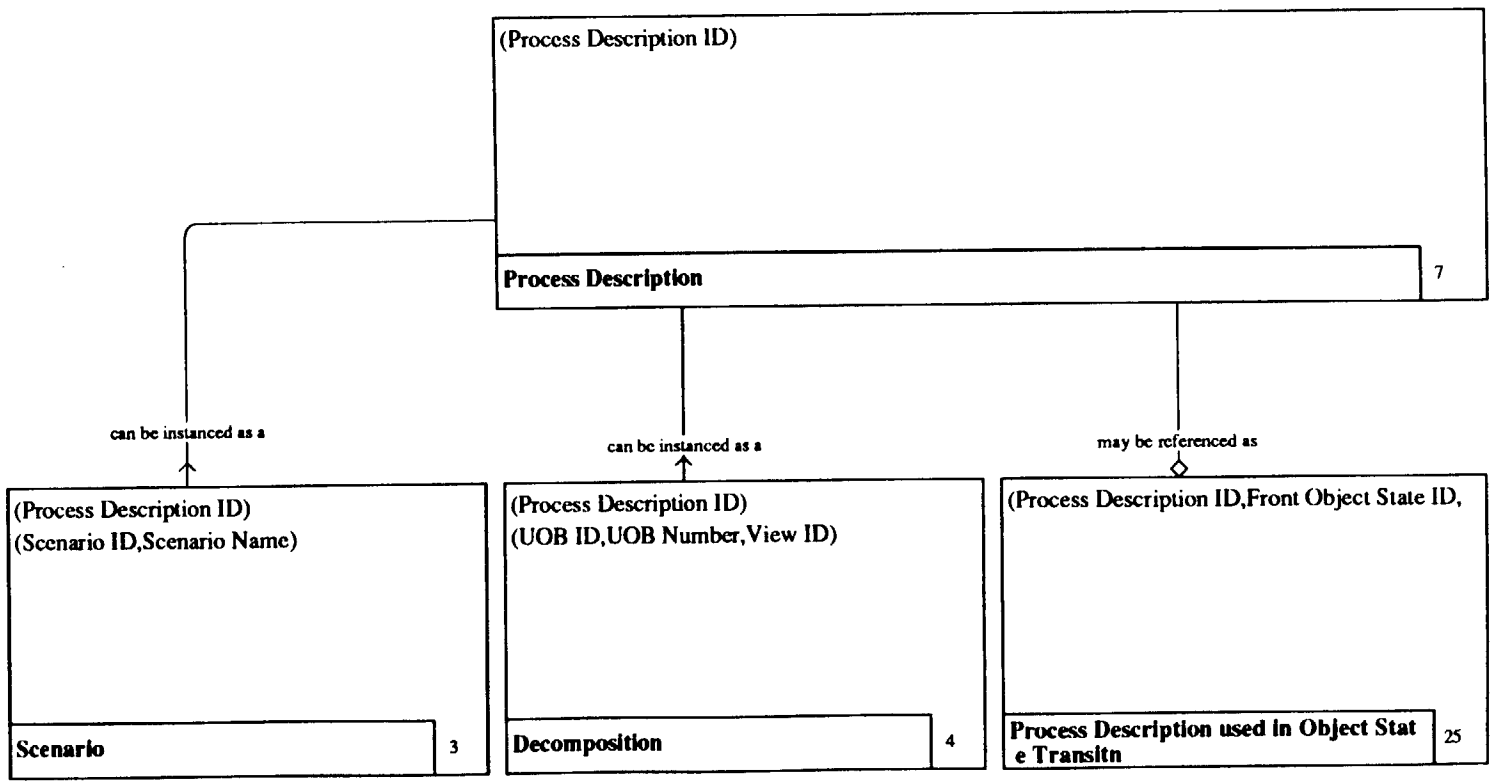
Name: View Glossary
Definition: A textual description given to a view to assist in the readership of the description.

Name: View ID
Definition: A unique identifier of a view. View names are also unique but this key is carried along to assist in the integration of multiple IDEF3 process descriptions. descriptions.

Name: View Name
Definition: The name associated with a view. This name appears on any decomposition which is defined relative to this view.

Name: View Type
Definition: In IDEF3 there are two types of decomposition views (objective and role) this attribute records which type a specific decomposition is.

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			

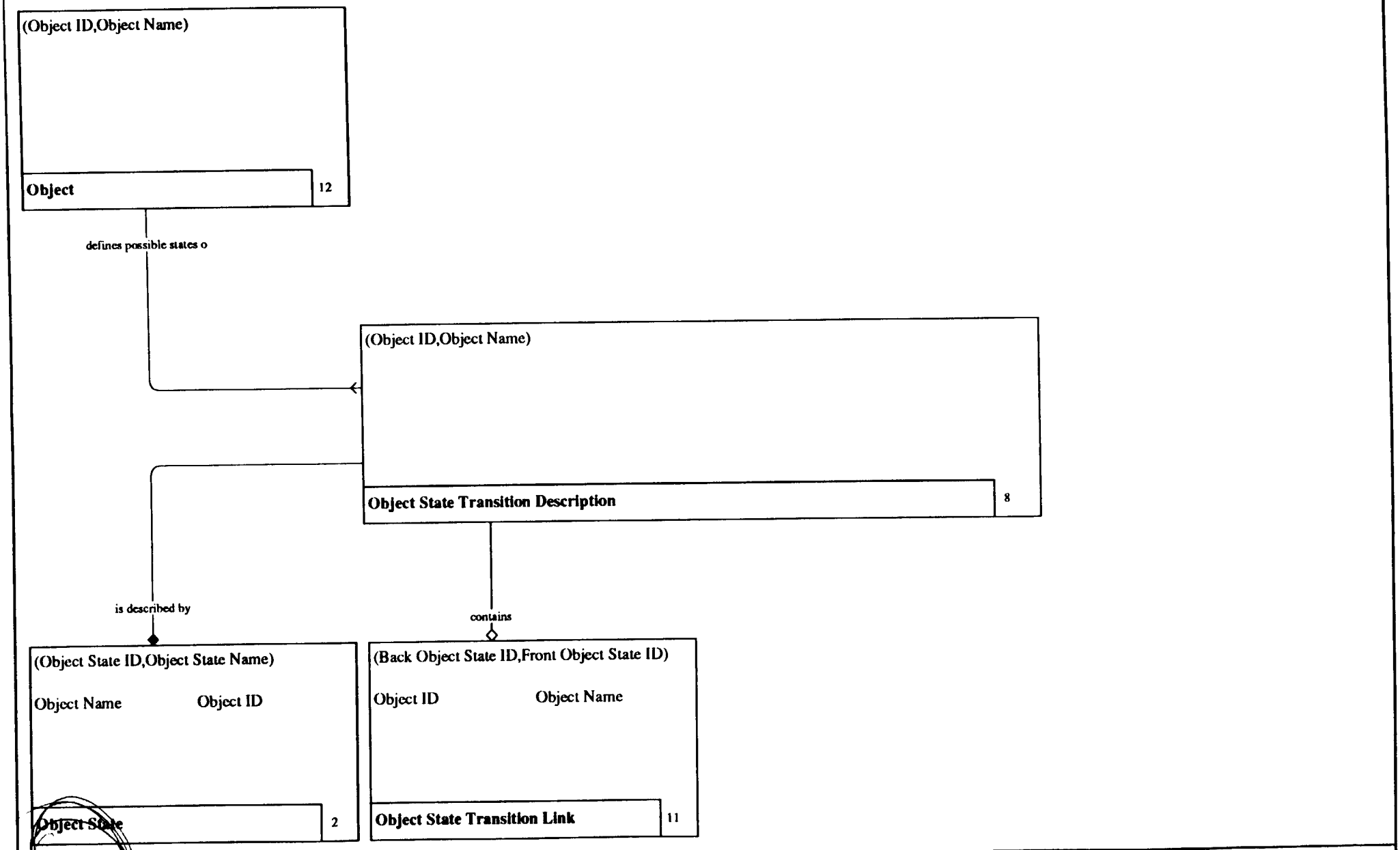


Node: E7	Title: Process Description	Number: 43

Entity Class Definition: Any collection of UOBs and associated links is a process description. This entity class tracks all of the scenarios and decompositions in an IDEF3 diagram.

Key Classes:
(Process Description ID)
Owned Attribute Classes:
Name: Process Description ID
Definition: The description unique identifier for any scenario or decomposition based description of a process in IDEF3.

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



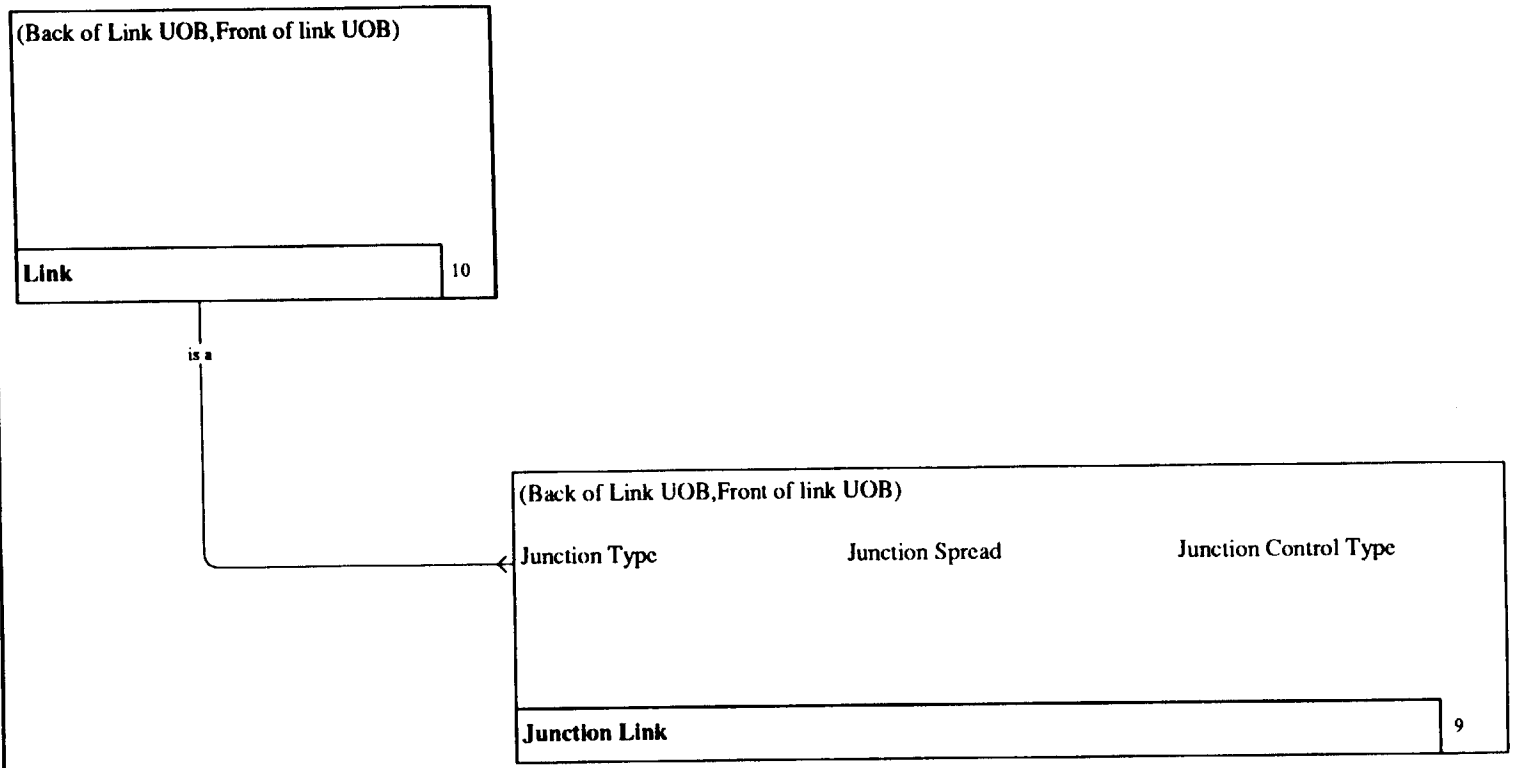
Node: E8	Title: Object State Transition Description	Number: 41

Entity Class Definition: The set of object states, transition links, and process description references that make up the description of the valid states that an object may exist in and the conditions for transitioning from state to state.

Key Classes:
(Object ID,Object Name)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Object Name	Object	12	Object	12	has an associated
Object ID	Object	12	Object	12	has an associated

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



Node: E9	Title: Junction Link	Number: 39

Entity Class Definition: A type of link between two UOBs that passes through one of the predefined junction types.

Key Classes:
(Back of Link UOB,Front of link UOB)
Owned Attribute Classes:

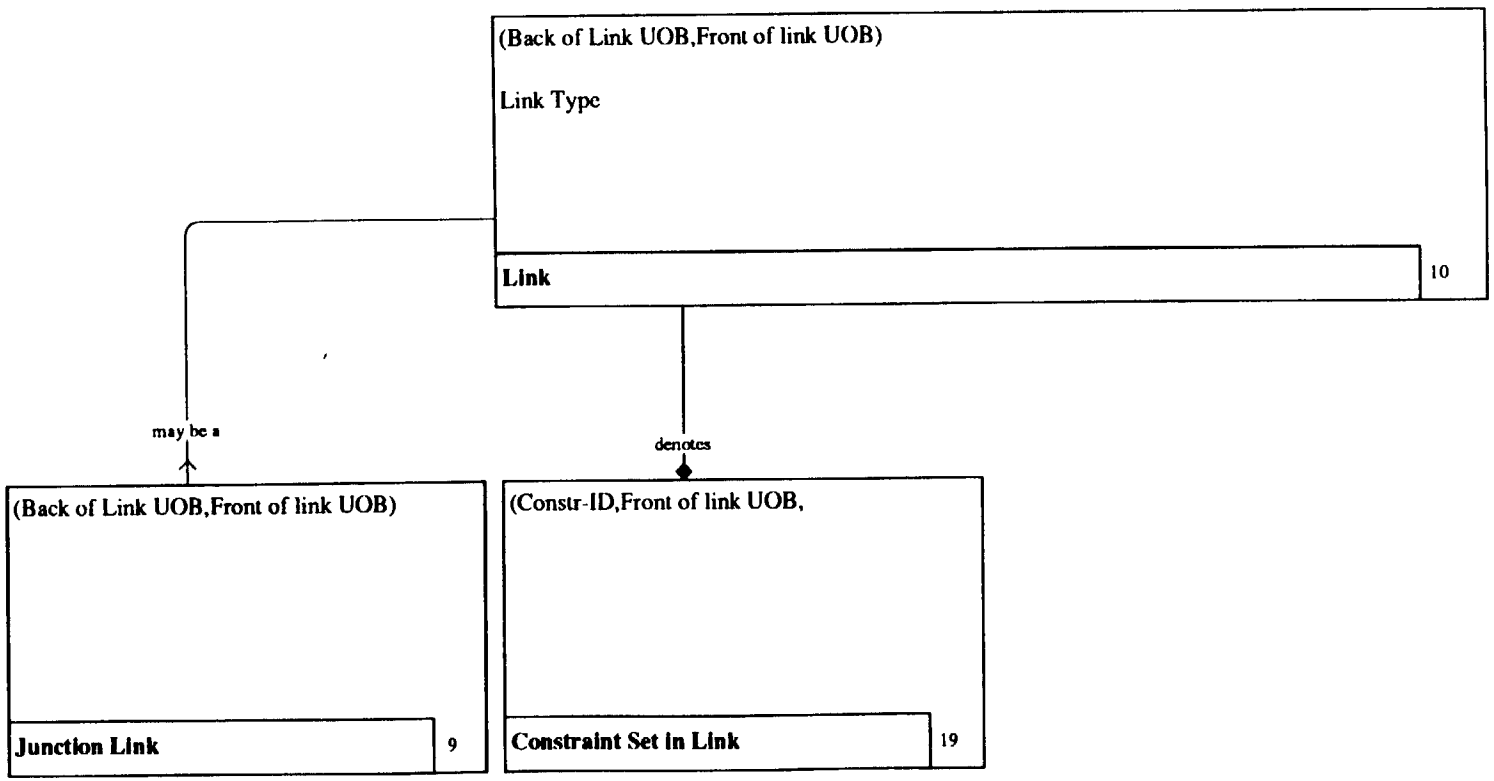
Name: Junction Control Type
Definition: Junctions can be used to represent synchronous or asyncronous control logic. This attribute captures the control type of a specific junction.

Name: Junction Spread
Definition: This attribute describes whether the junction is on the front of a link set or on the back of a link set.

Name: Junction Type
Definition: There are three basic junction types: 1) AND junctions 2) OR junctions 3) XOR junctions This attribute records which of these types a particular junction is.

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Front of link UOB	Link	10	Link	10	may be a
Back of Link UOB	Link	10	Link	10	may be a

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



Entity Class Definition: This entity class records the information about distinguished relations between UOBs that are denoted by links on an IDEF3 diagram.

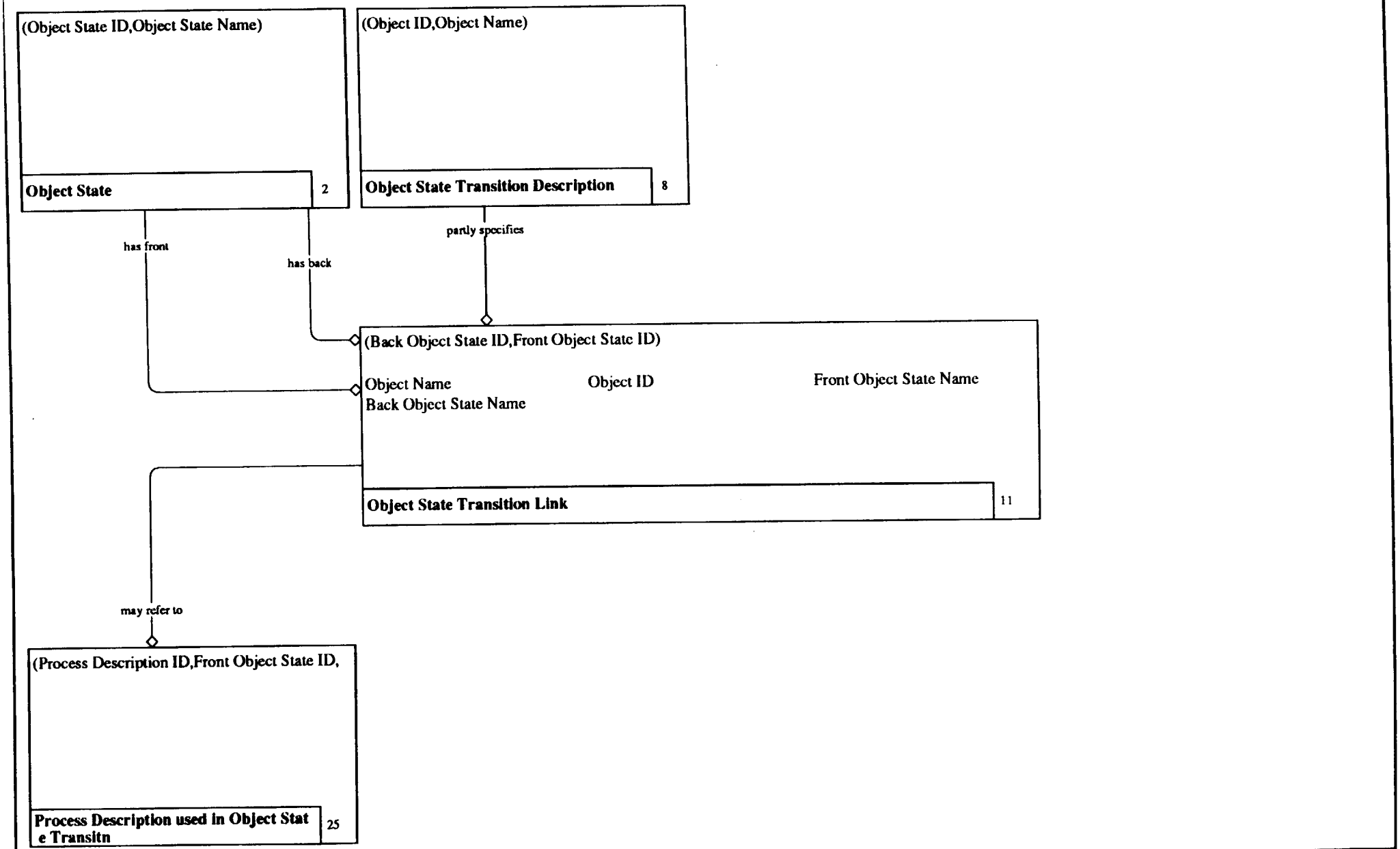
Key Classes:
(Back of Link UOB,Front of link UOB)
Owned Attribute Classes:

Name: Back of Link UOB
Definition: This attribute records the UOB that is at the termination of a link.

Name: Front of link UOB
Definition: This attribute captures which UOB is at the front of a particular link.

Name: Link Type
Definition: There are three built-in link types: 1) Precidence Links 2) Object Flow Links 3) Relational Links This attribute captures the specification of which type a particular link is.

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



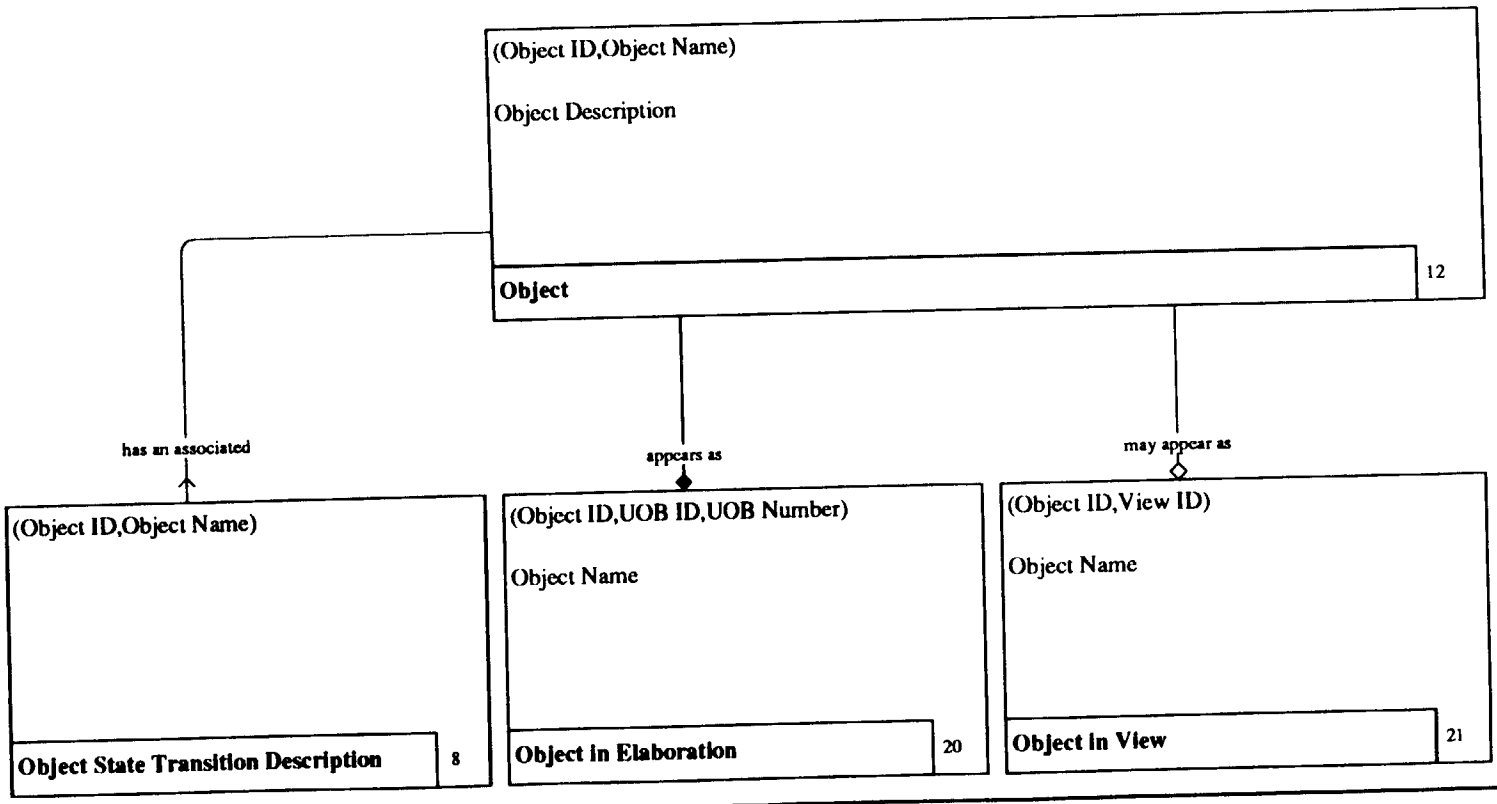
Node: E11	Title: Object State Transition Link	Number: 35

Entity Class Definition: The denotation of the conditions necessary and sufficient for an object to change states.

Key Classes:
(Back Object State ID,Front Object State ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Back Object State Name	Object State	2	Object State	2	is back of
Back Object State ID	Object State	2	Object State	2	is back of
Front Object State Name	Object State	2	Object State	2	is front of
Front Object State ID	Object State	2	Object State	2	is front of
Object Name	Object	12	Object State Transition Description	8	contains
Object ID	Object	12	Object State Transition Description	8	contains

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



Node: E12	Title: Object	Number: 33

Entity Class Definition: This entity class records the information in IDEF3 about a specific object (real or conceptual) or class/type of object which is a part of a process description, Object state transition, or UOB elaboration.

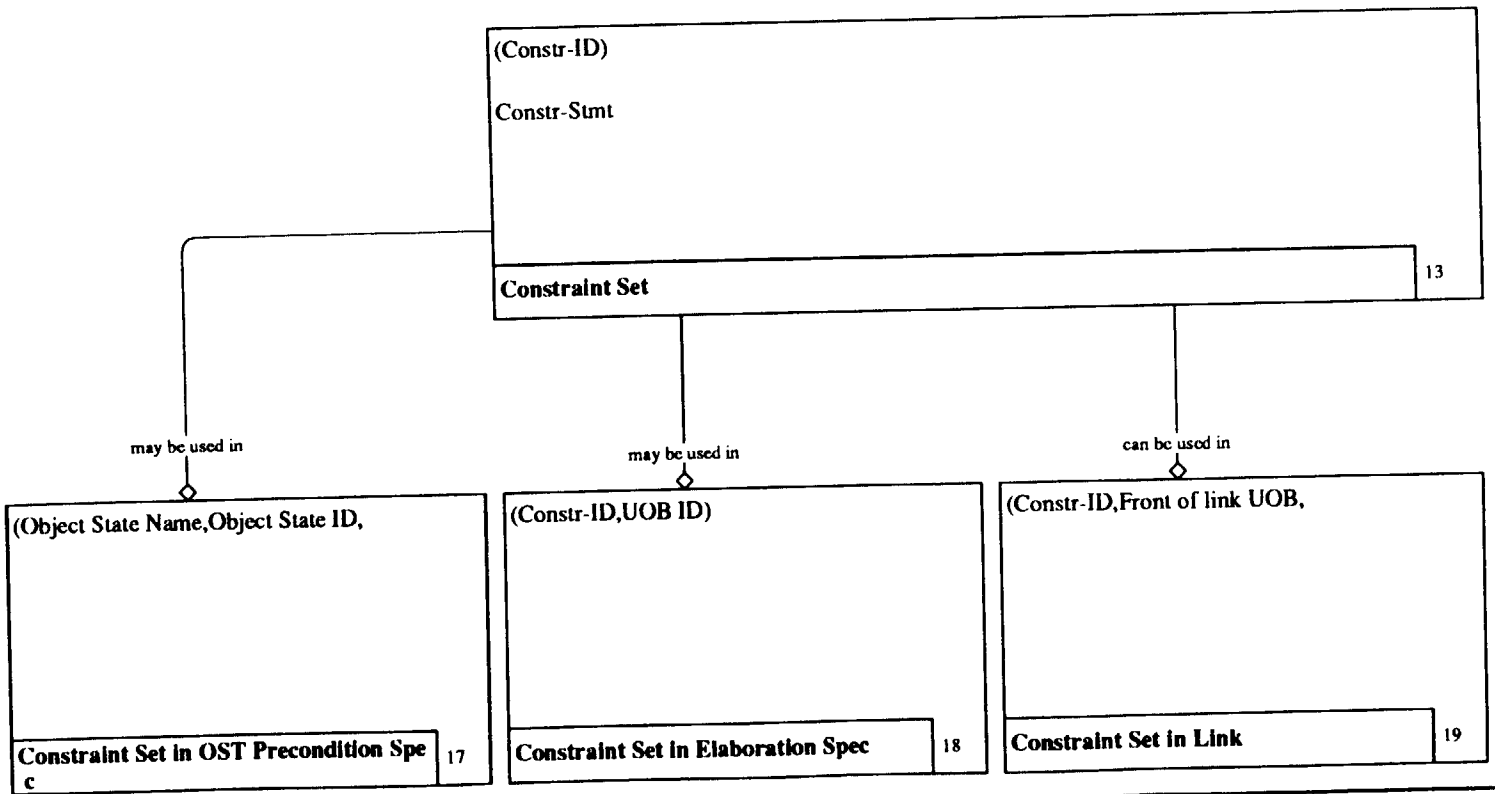
Key Classes:
(Object ID,Object Name)
Owned Attribute Classes:

Name: Object Description
Definition: A textual glossary associated with an object type.

Name: Object ID
Definition: A unique identifier given to an object type to assist in the integration of multiple IDEF3 descriptions.

Name: Object Name
Definition: The unique name of an object type.

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



Node: E13	Title: Constraint Set	Number: 31

Entity Class Definition: Each link must have 1 or many constraint specifications associated with it. Each junction may also have constraint sets associated with them that add additional specifications to the standard control logic implied by the junction type. Finally each elaboration has a constraint set associated with it.

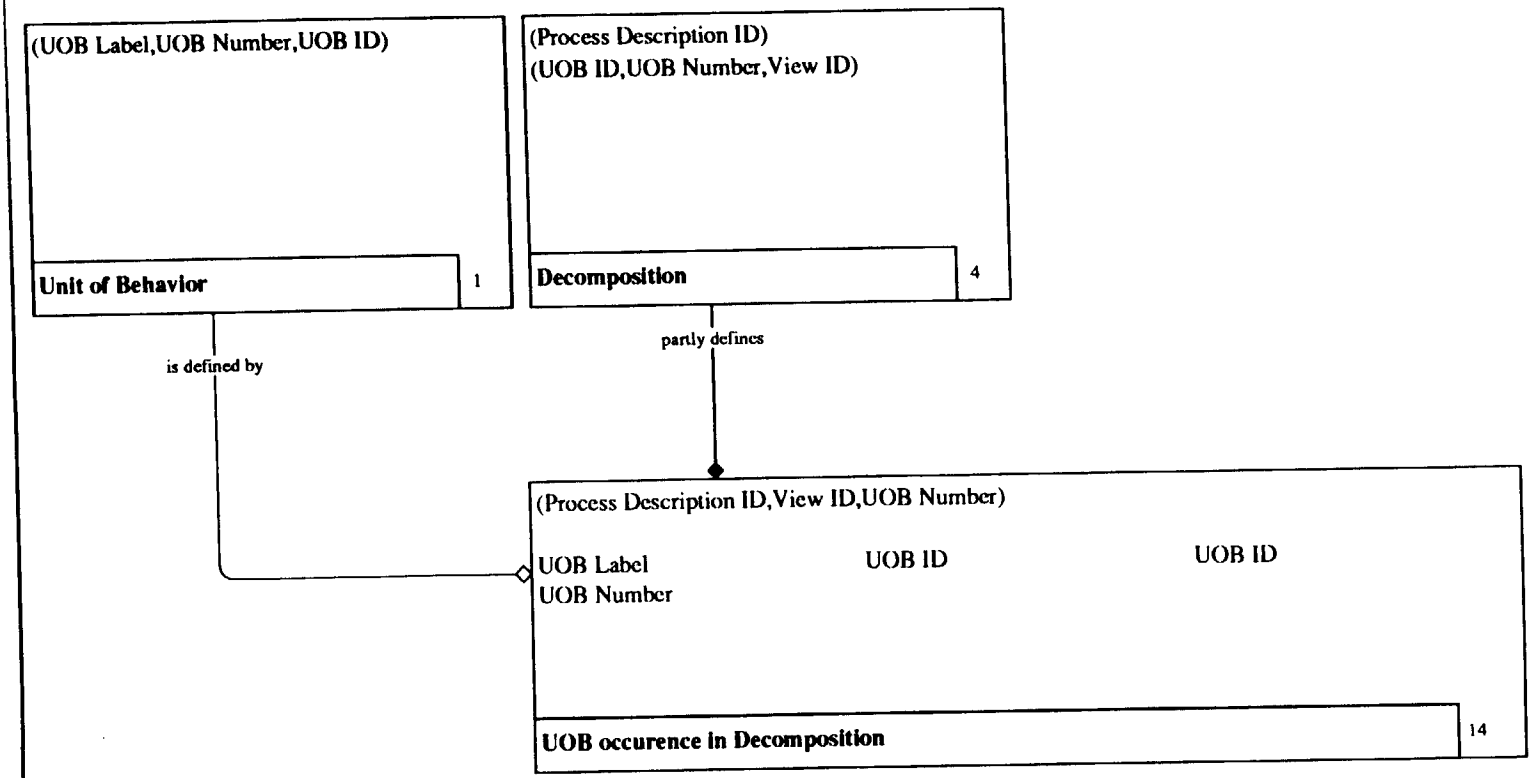
Key Classes:
(Constr-ID)
Owned Attribute Classes:

Name: Constr-ID
Definition: The unique identifier of a constraint specification.

Name: Constr-Stmt
Definition: A textual statement of a constraint. In future versions of the IDEF3 this would be specified in a version of the IISyCL language.

Used At:	Author: Tom Blinn	Date: 11-19-89			
	Project: IDEF32	Rev:	Working	READER	DATE
	Notes: 1 2 3 4 5 6 7 8 9 10		Draft		
			Recommended		
			Publication		

Context



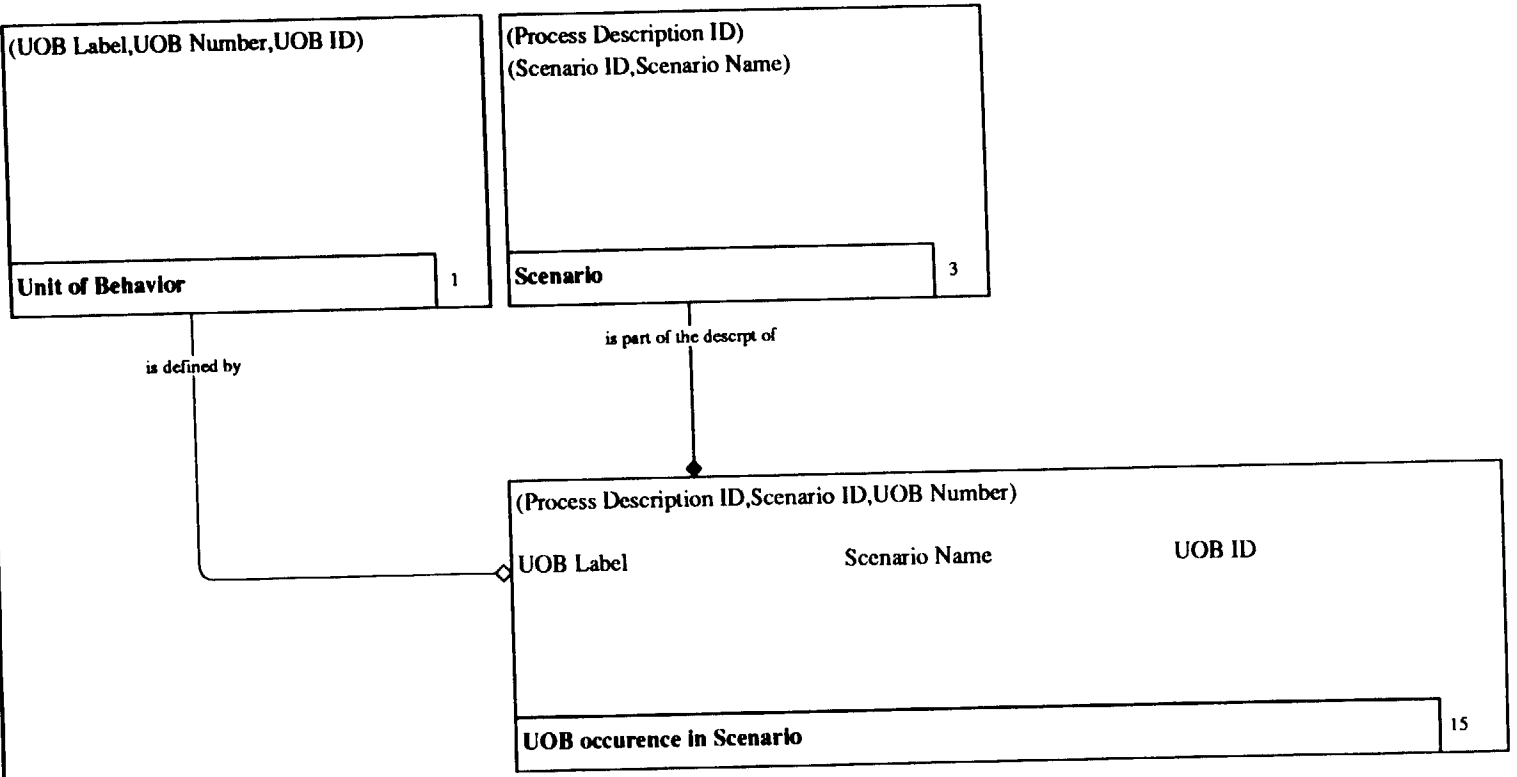
Node: E14	Title: UOB occurence in Decomposition	Number: 29

Entity Class Definition: A UOB can occur many times in many decompositions. This entity class tracks each of these occurences.

Key Classes:
(Process Description ID,View ID,UOB Number)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Process Description ID	Process Description	7	Decomposition	4	is described by
View ID	View	6	Decomposition	4	is described by
UOB Number	Unit of Behavior	1	Decomposition	4	is described by
UOB ID	Unit of Behavior	1	Decomposition	4	is described by
UOB ID	Unit of Behavior	1	Unit of Behavior	1	can be used in many
UOB Number	Unit of Behavior	1	Unit of Behavior	1	can be used in many
UOB Label	Unit of Behavior	1	Unit of Behavior	1	can be used in many

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



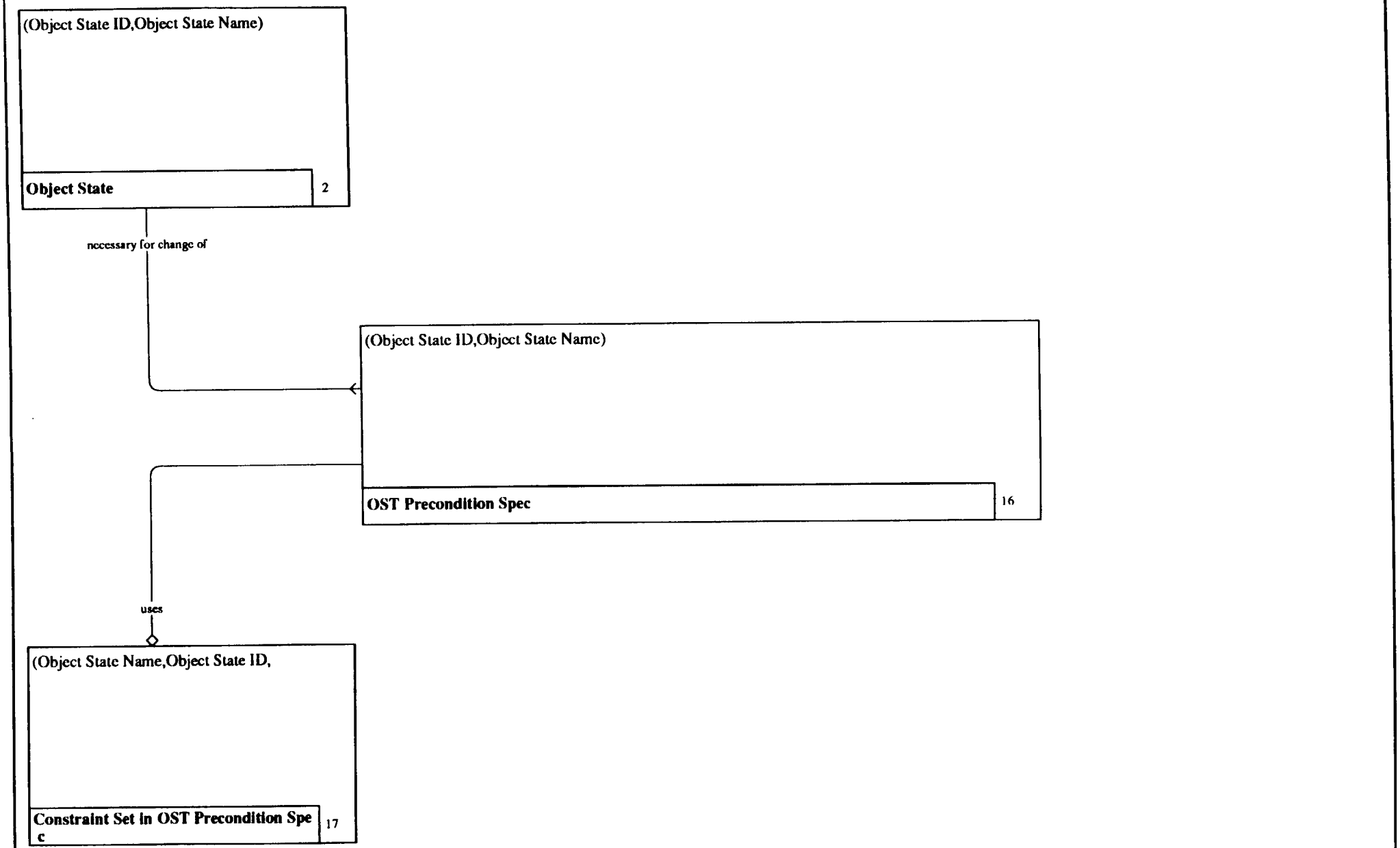
Node: E15	Title: UOB occurence in Scenario	Number: 21

Entity Class Definition: A UOB can be a part of many scenario descriptions. This entity class tracks each of those occurences.

Key Classes:
(Process Description ID,Scenario ID,UOB Number)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Process Description ID	Process Description	7	Scenario	3	is described by the
UOB ID	Unit of Behavior	1	Unit of Behavior	1	can be used in
Scenario Name	Scenario	3	Scenario	3	is described by the
Scenario ID	Scenario	3	Scenario	3	is described by the
UOB Number	Unit of Behavior	1	Unit of Behavior	1	can be used in
UOB Label	Unit of Behavior	1	Unit of Behavior	1	can be used in

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



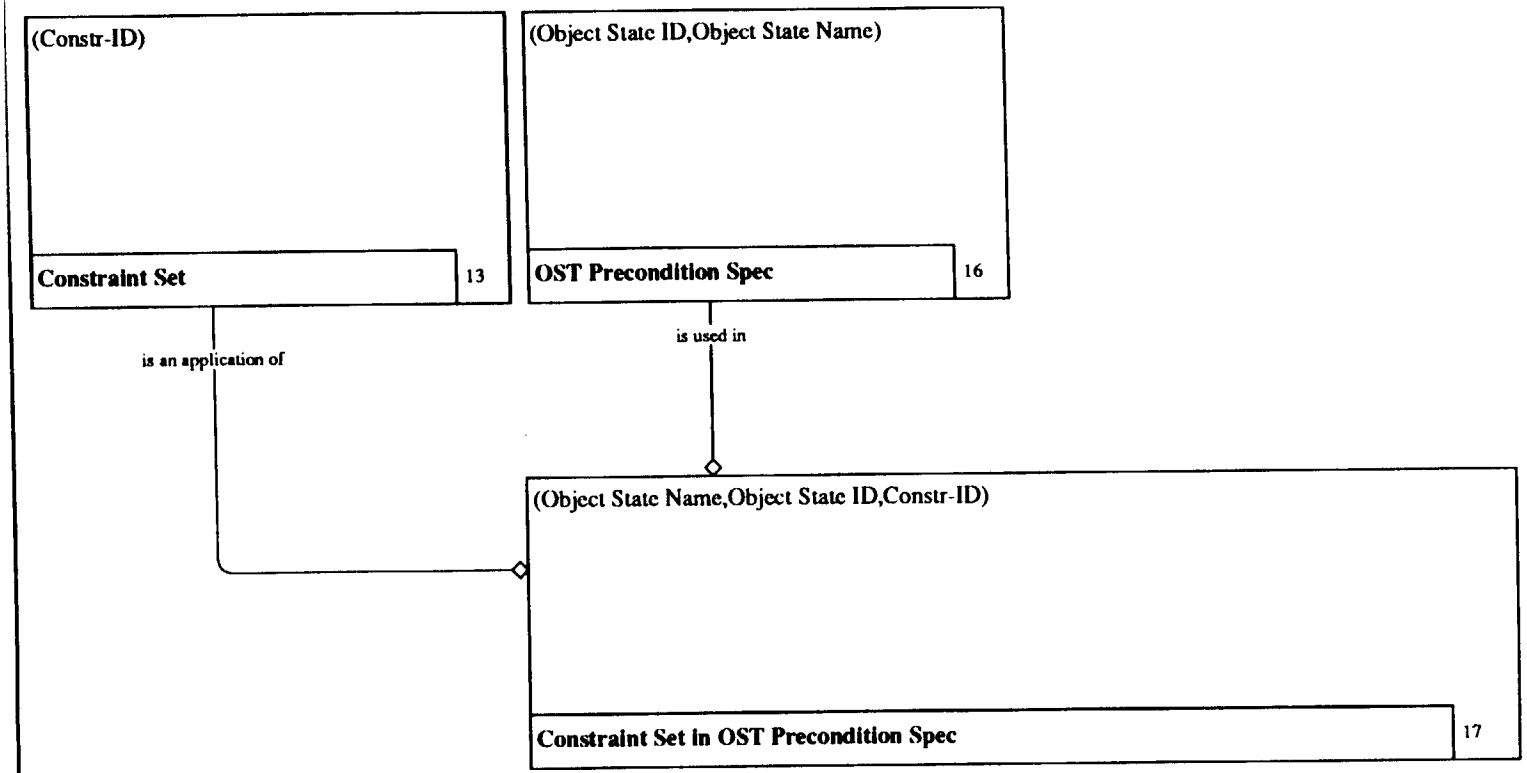
Node: E16	Title: OST Precondition Spec	Number: 25

Entity Class Definition: The conditions that must be satisfied for an object state transition to be enabled. These are necessary conditions but not sufficient conditions. conditions.

Key Classes:
(Object State ID,Object State Name)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path		
			Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Object State Name	Object State	2	Object State	2	has
Object State ID	Object State	2	Object State	2	has

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



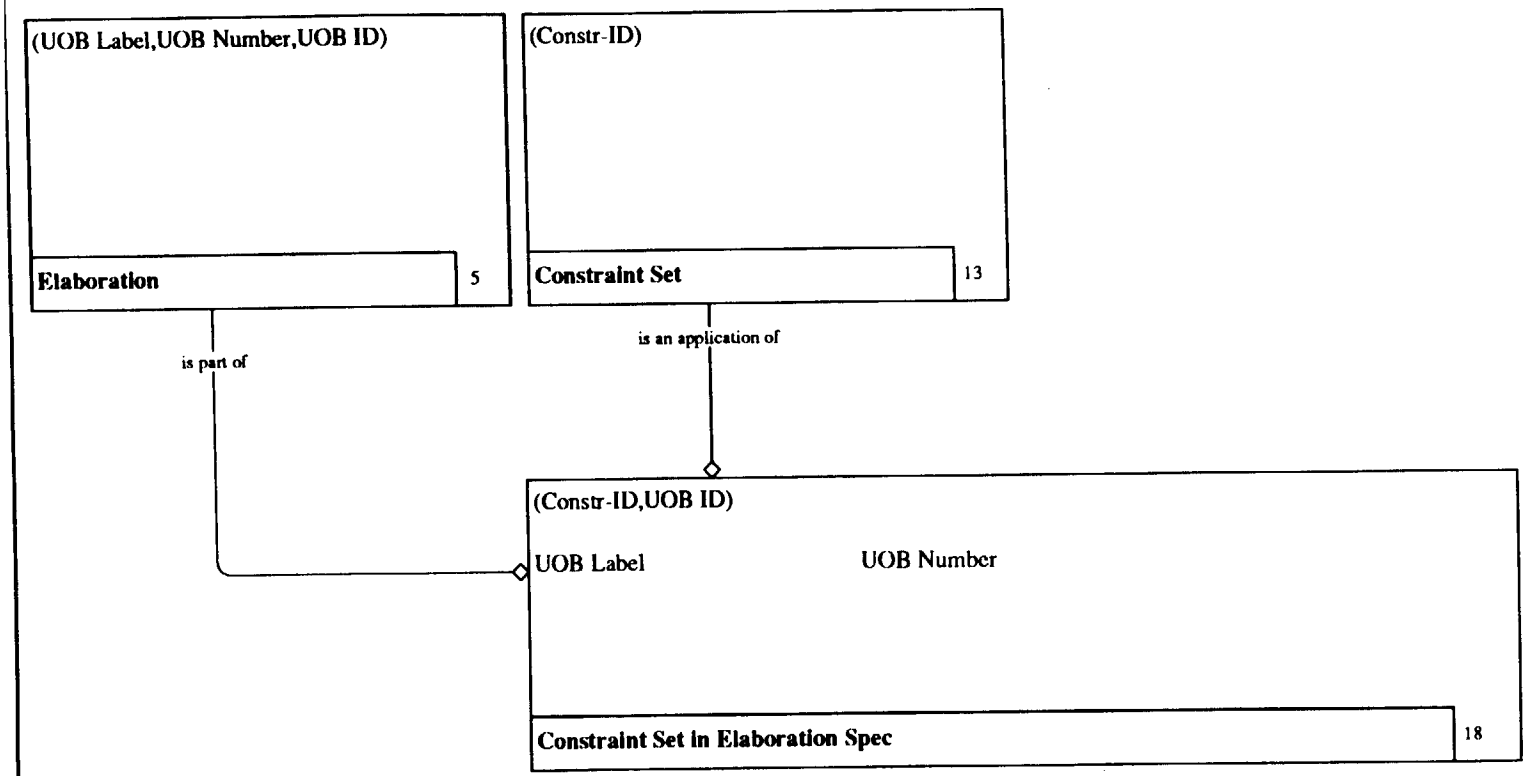
Node: E17	Title: Constraint Set in OST Precondition Spec	Number: 23

Entity Class Definition: The use of a constraint set in an Object State Transition Precondition specification.

Key Classes:
(Object State Name,Object State ID,Constr-ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Object State Name	Object State	2	OST Precondition Spec	16	uses
Object State ID	Object State	2	OST Precondition Spec	16	uses
Constr-ID	Constraint Set	13	Constraint Set	13	may be used in

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



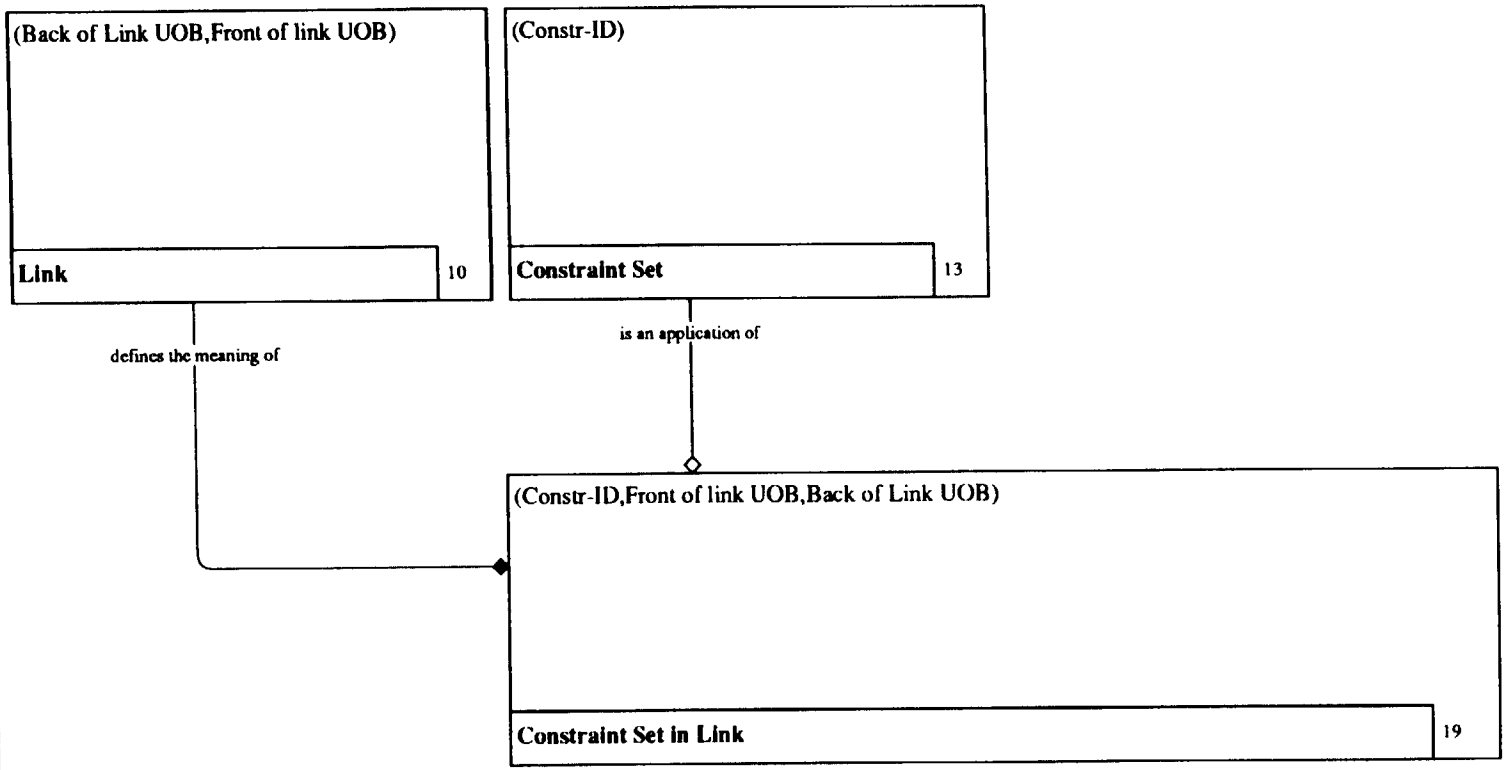
Node: E18	Title: Constraint Set in Elaboration Spec	Number: 21

Entity Class Definition: The use of a constraint set as a part of an elaboration specification.

Key Classes:
(Constr-ID,UOB ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Constr-ID	Constraint Set	13	Constraint Set	13	may be used in
UOB ID	Unit of Behavior	1	Elaboration	5	calls for use of
UOB Number	Unit of Behavior	1	Elaboration	5	calls for use of
UOB Label	Unit of Behavior	1	Elaboration	5	calls for use of

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



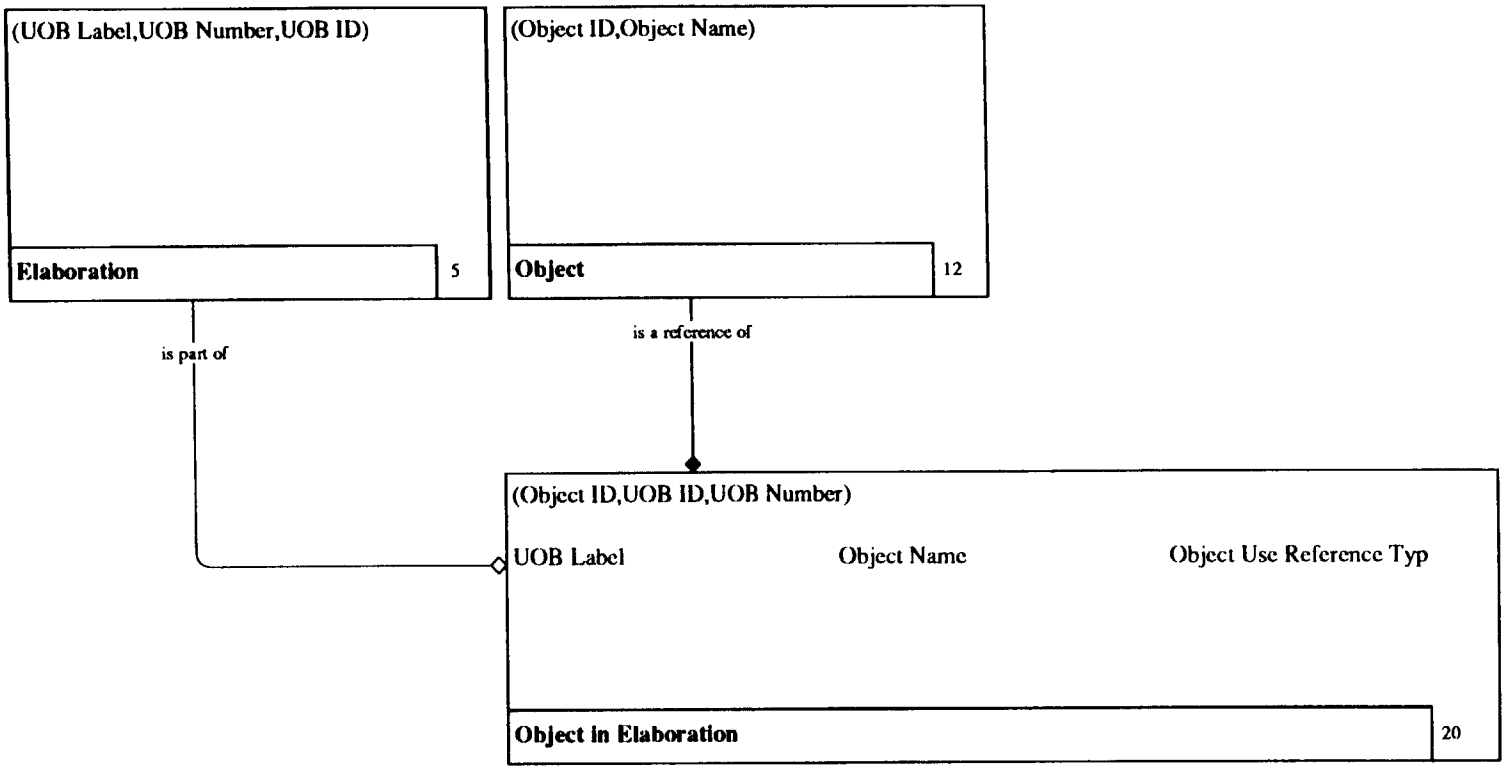
Node: E19	Title: Constraint Set in Link	Number: 19

Entity Class Definition: The use of a constraint set in a link between two UOBs.

Key Classes:
(Constr-ID,Front of link UOB,Back of Link UOB)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Constr-ID	Constraint Set	13	Constraint Set	13	can be used in
Front of link UOB	Link	10	Link	10	denotes
Back of Link UOB	Link	10	Link	10	denotes

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



Node: E20	Title: Object in Elaboration	Number: 17

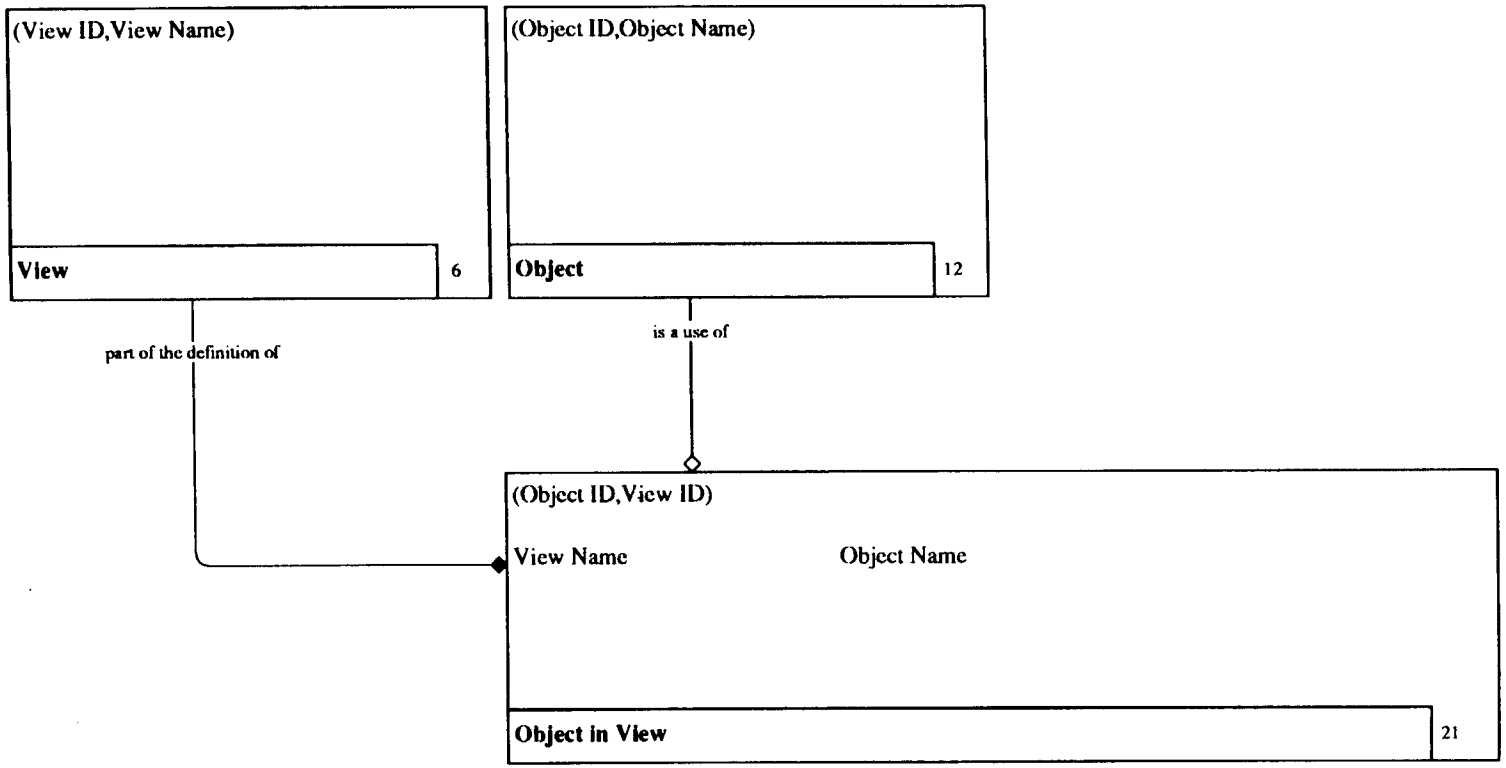
Entity Class Definition: An object can be referenced in many elaborations. This entity records such a reference and the specificity of that reference.

Key Classes:
(Object ID,UOB ID,UOB Number)
Owned Attribute Classes:

Name: Object Use Reference Typ
Definition: This attribute records the use of an object reference as either specific (refering to a specific object) or non-specific (refering to an instance of type of object). The latter reference is often called an "indeterminate" reference.

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Object Name	Object	12	Object	12	appears as
Object ID	Object	12	Object	12	appears as
UOB ID	Unit of Behavior	1	Elaboration	5	has
UOB Number	Unit of Behavior	1	Elaboration	5	has
UOB Label	Unit of Behavior	1	Elaboration	5	has

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



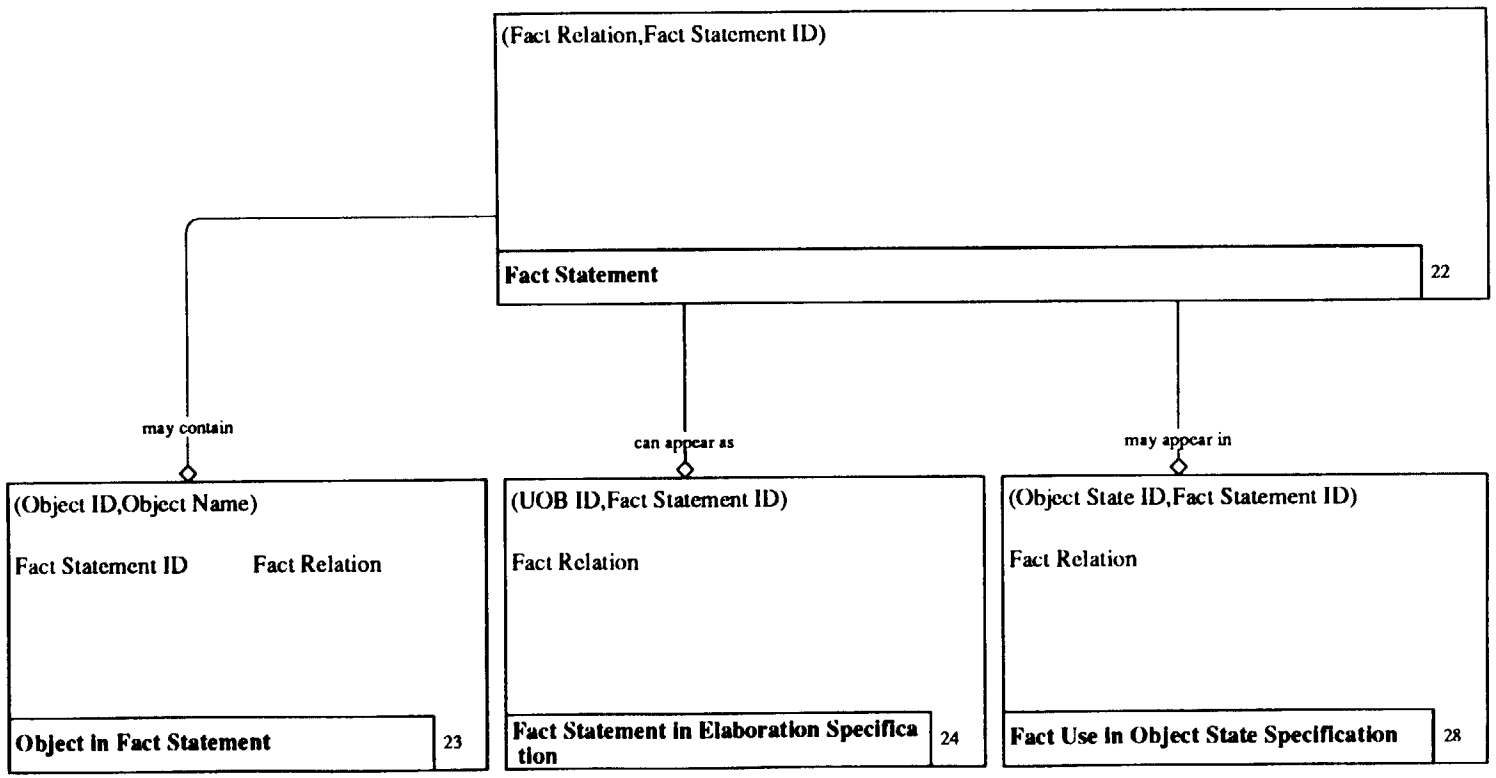
Node: E21	Title: Object in View	Number: 15

Entity Class Definition: An object can be used to establish many views. This entity records the use of an object reference in a particular view.

Key Classes:
(Object ID,View ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Object Name	Object	12	Object	12	may appear as
Object ID	Object	12	Object	12	may appear as
View Name	View	6	View	6	is defined by
View ID	View	6	View	6	is defined by

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



Node: E22	Title: Fact Statement	Number: 13

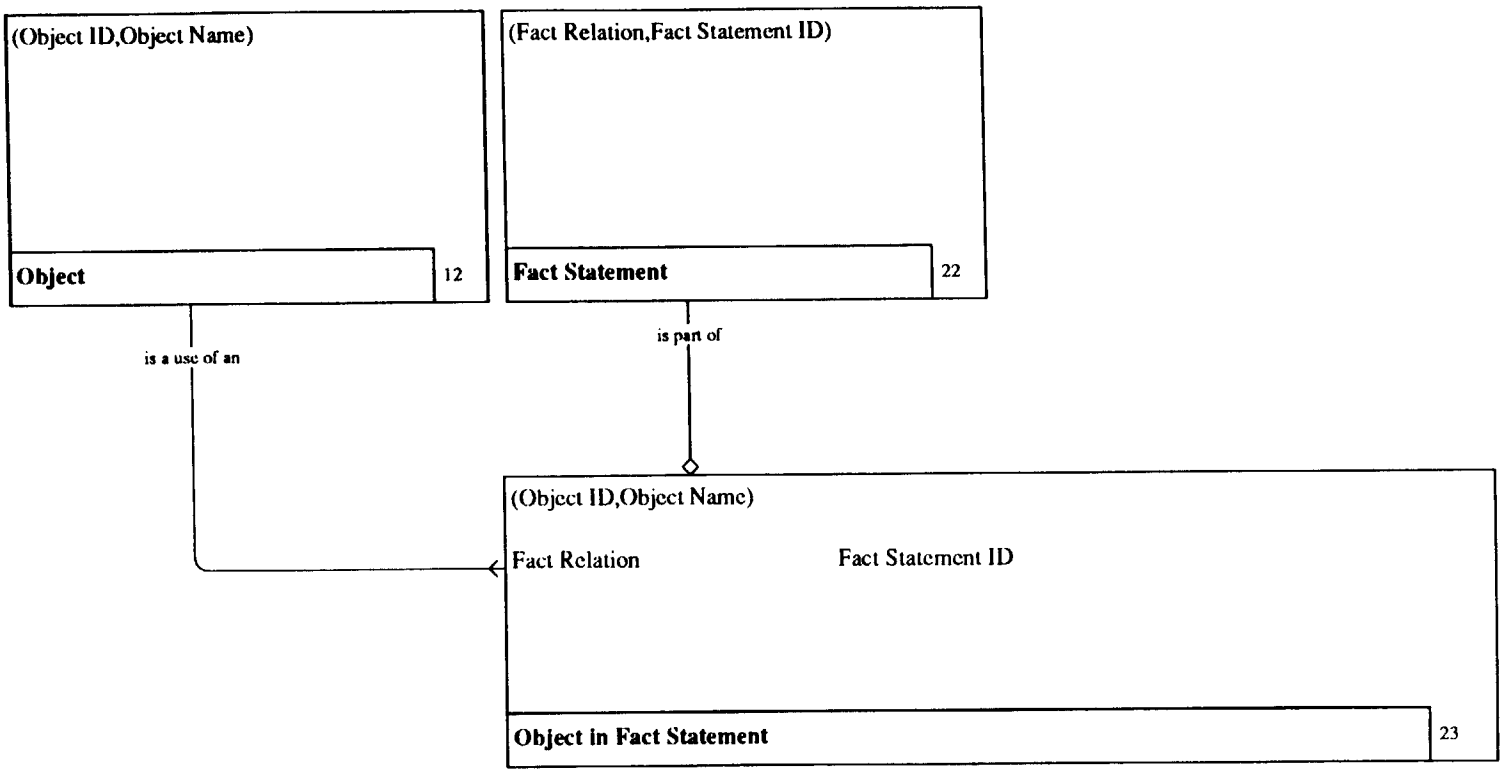
Entity Class Definition: A proposition with a set of objects and a relationship and a polarity.

Key Classes:
(Fact Relation,Fact Statement ID)
Owned Attribute Classes:

Name: Fact Relation
Definition: The relation used to form the proposition over the object set.

Name: Fact Statement ID
Definition: A unique identifier given to a fact statement.

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



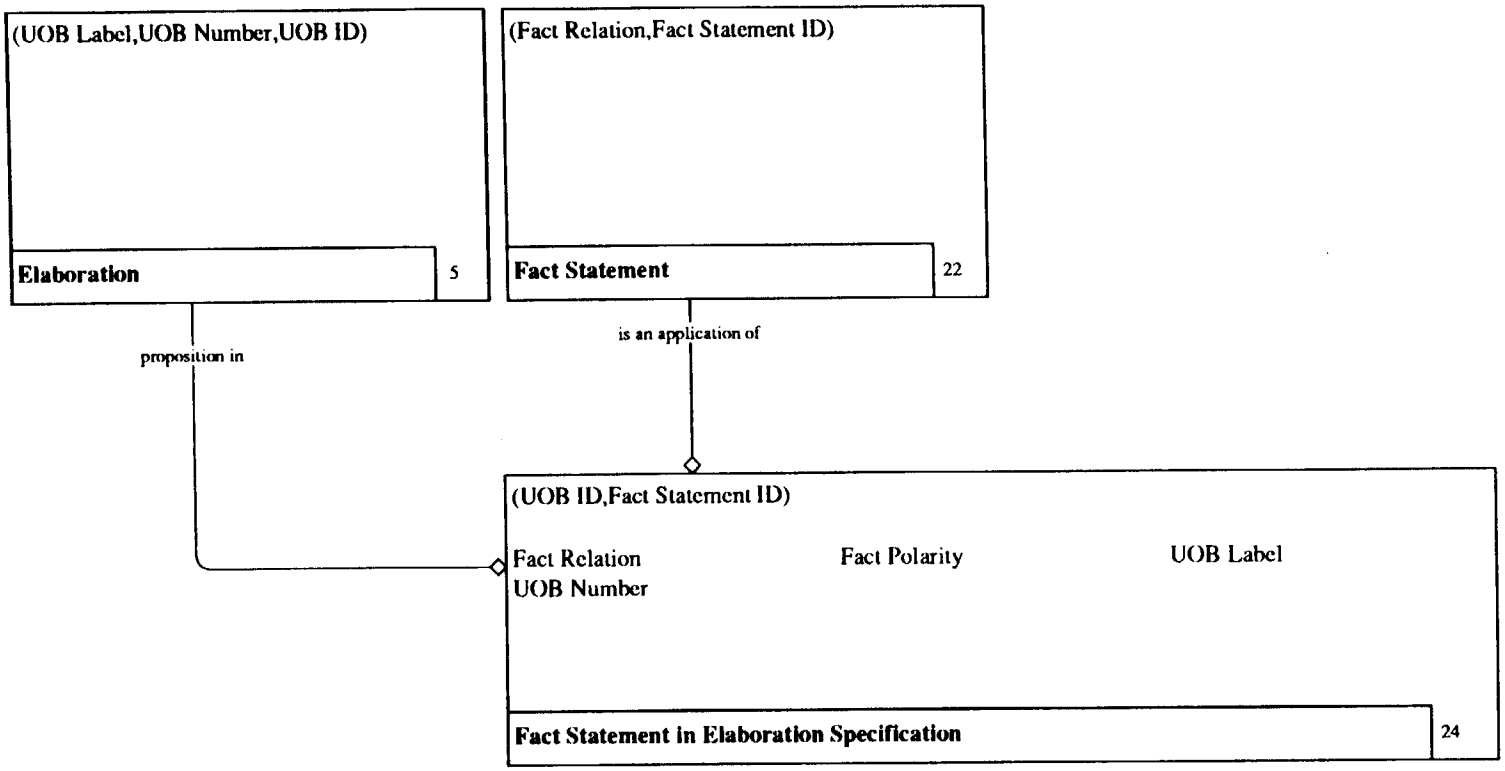
Node: E23	Title: Object in Fact Statement	Number: 11

Entity Class Definition: The reference of an object type in a proposition as a part of an elaboration.

Key Classes:
(Object ID,Object Name)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Object Name	Object	12	Object	12	may appear as
Object ID	Object	12	Object	12	may appear as
Fact Statement ID	Fact Statement	22	Fact Statement	22	may contain
Fact Relation	Fact Statement	22	Fact Statement	22	may contain

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



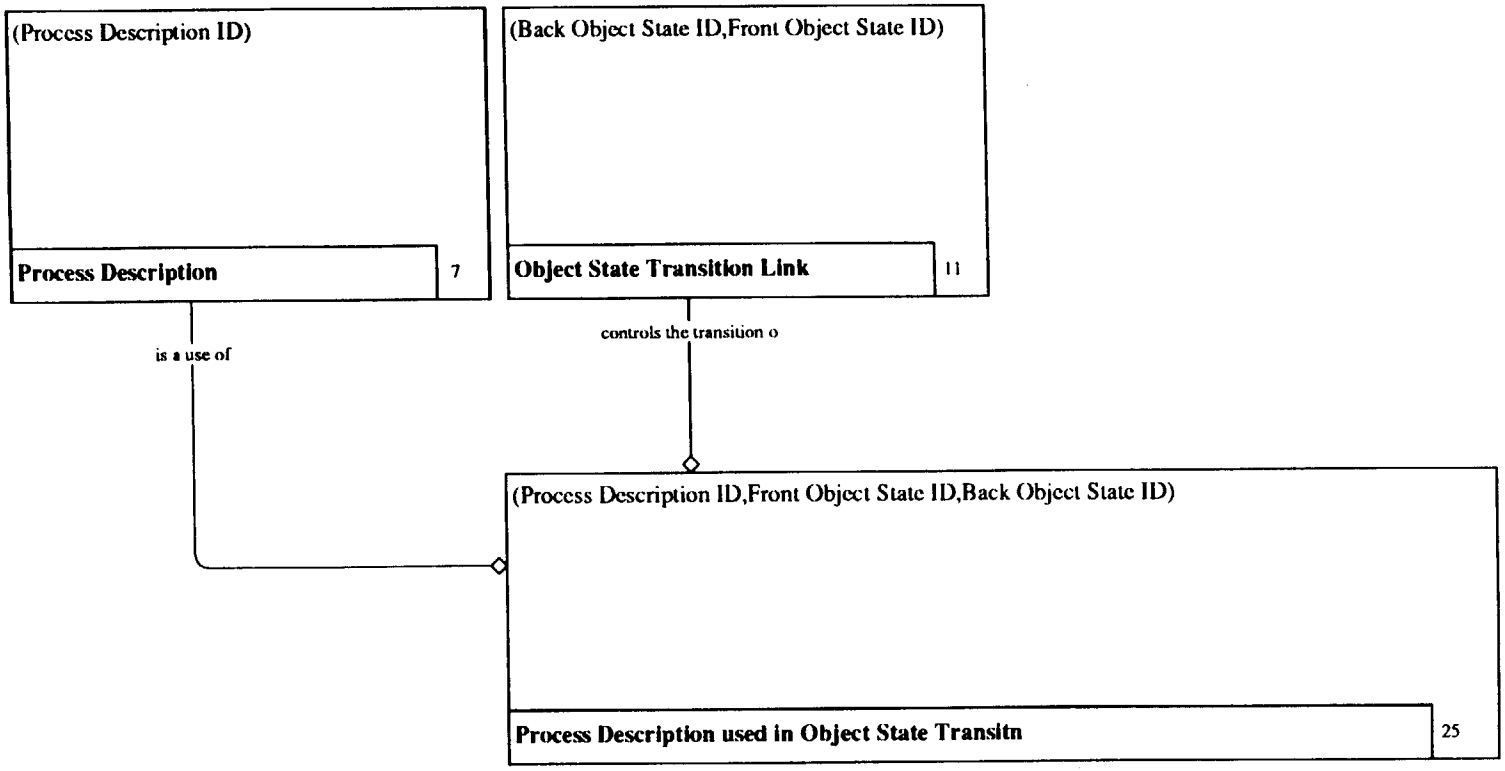
Node: E24	Title: Fact Statement in Elaboration Specification	Number: 9

Entity Class Definition: The use of a fact statement in a UOB elaboration specificatoin.

Key Classes:
(UOB ID,Fact Statement ID)
Owned Attribute Classes:
Name: Fact Polarity
Definition: The test criteria for determination of the truth value of a proposition associated with a fact statement used in a particular context.

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
UOB ID	Unit of Behavior	1	Elaboration	5	calls for the use of
UOB Number	Unit of Behavior	1	Elaboration	5	calls for the use of
UOB Label	Unit of Behavior	1	Elaboration	5	calls for the use of
Fact Statement ID	Fact Statement	22	Fact Statement	22	can appear as
Fact Relation	Fact Statement	22	Fact Statement	22	can appear as

Used At:	Author: Tom Blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF32	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



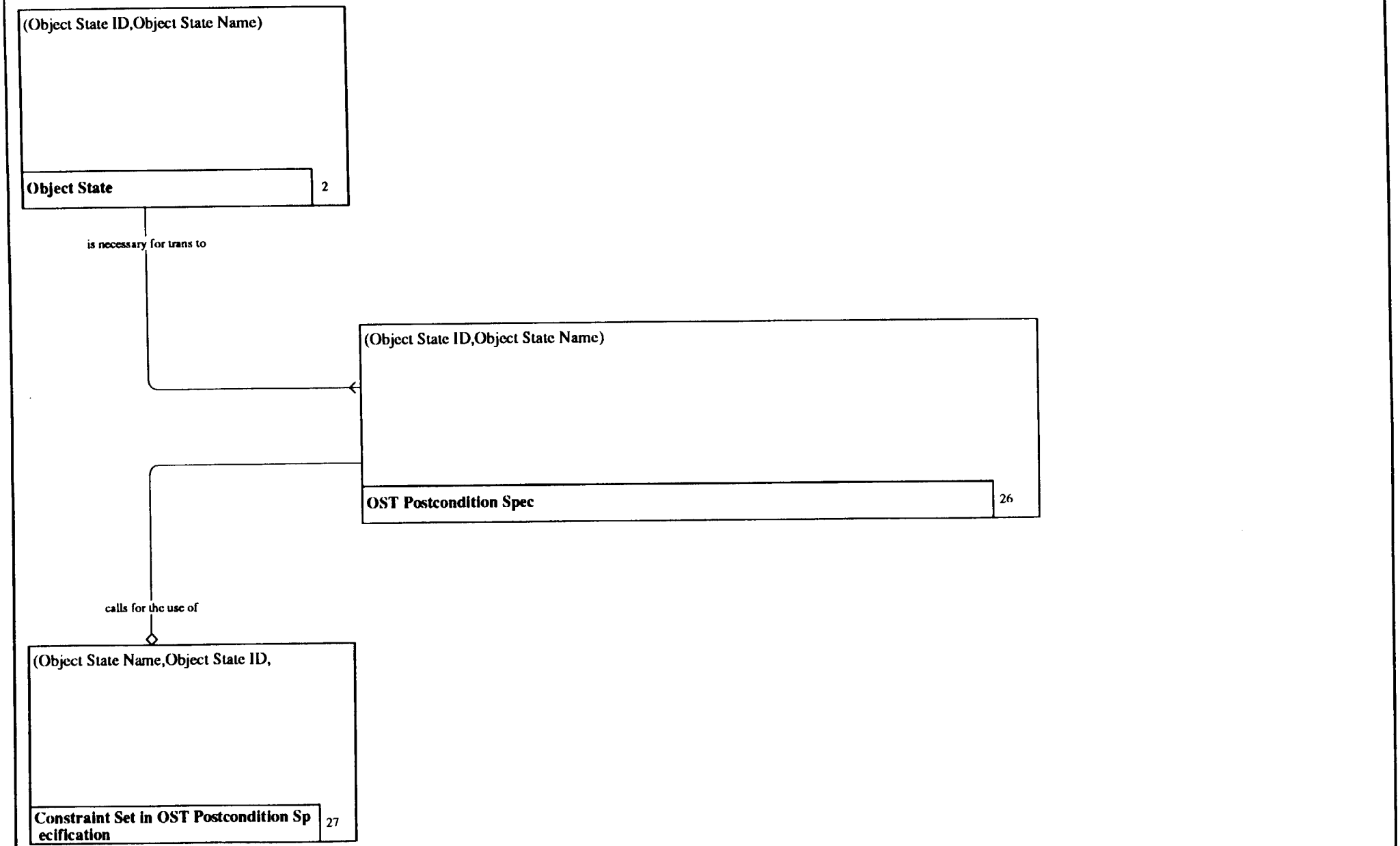
Node: E25	Title: Process Description used in Object State Transitn	Number: 7

Entity Class Definition: The reference of a process description as a necessary condition for an object state transition.

Key Classes:
(Process Description ID,Front Object State ID,Back Object State ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Process Description ID	Process Description	7	Process Description	7	may be referenced as
Front Object State ID	Object State	2	Object State Transition Link	11	may refer to
Back Object State ID	Object State	2	Object State Transition Link	11	may refer to

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



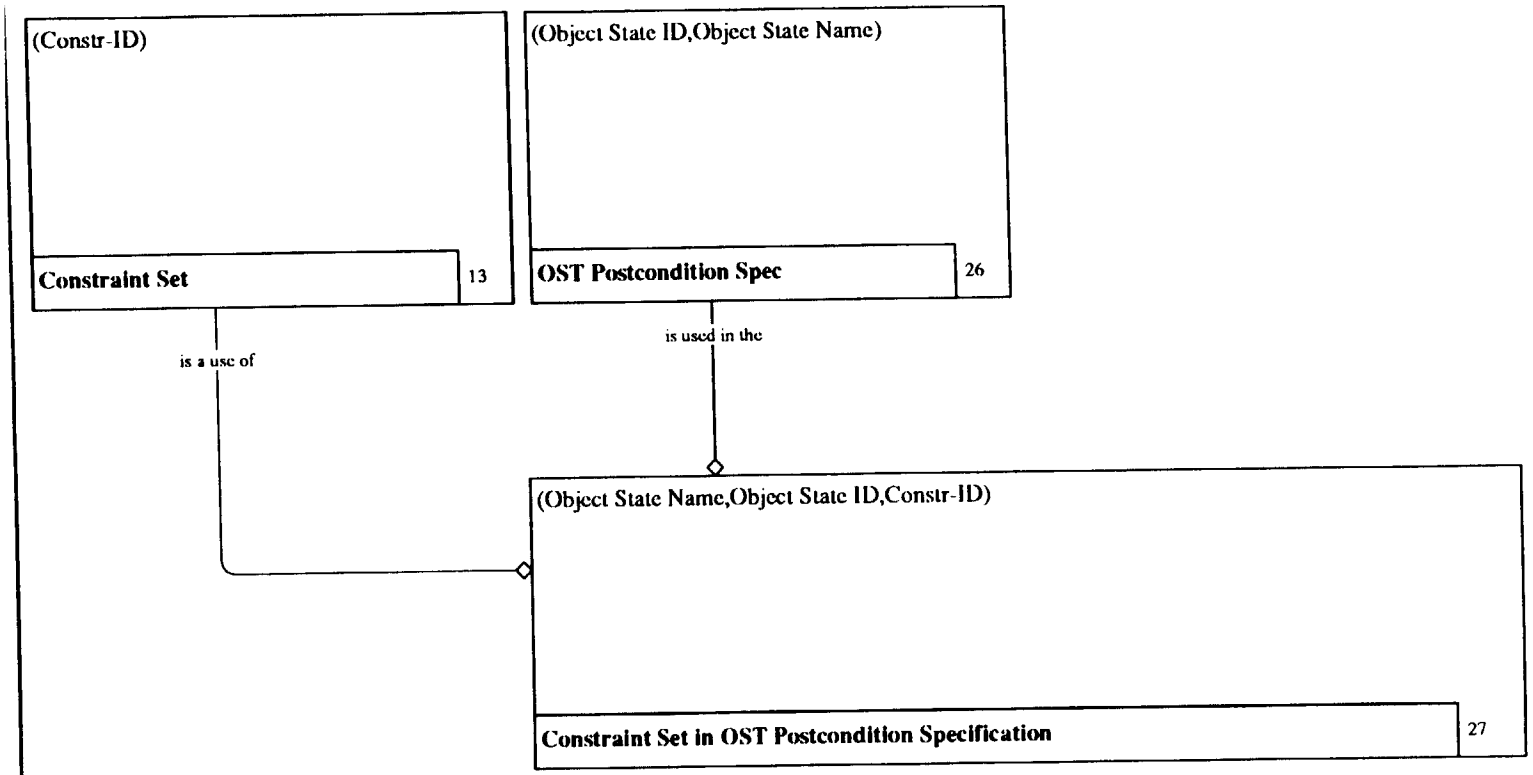
Node: E26	Title: OST Postcondition Spec	Number: 5

Entity Class Definition: This entity class manages the information about the conditions underwhich an object can complete the transition from one object state to another. another.

Key Classes:
(Object State ID,Object State Name)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path		
			Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Object State Name	Object State	2	Object State	2	has
Object State ID	Object State	2	Object State	2	has

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



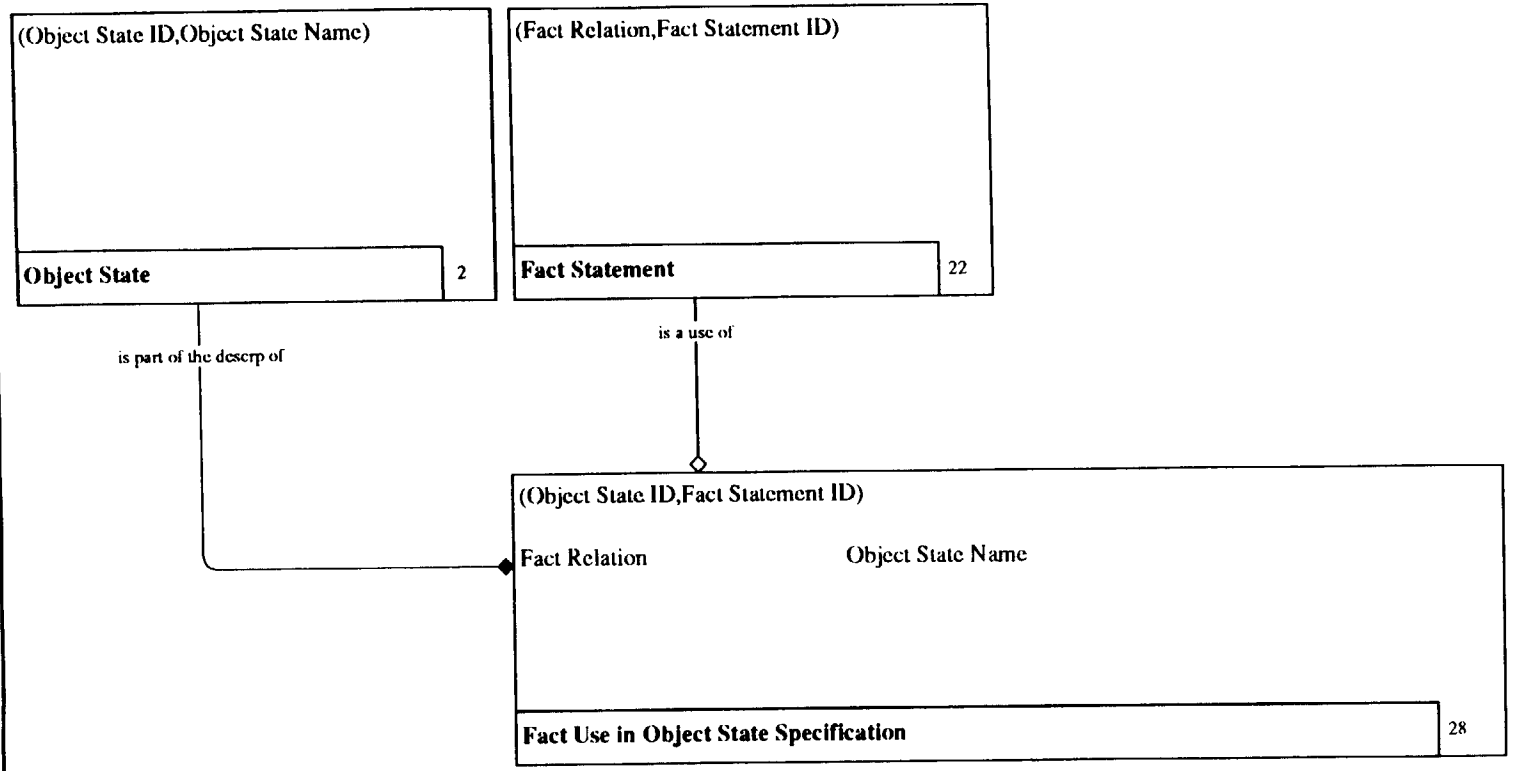
Node: E27	Title: Constraint Set in OST Postcondition Specification	Number: 3

Entity Class Definition: The use of a constraint set in the post conditions associated with an object state transition from one state to another.

Key Classes:
(Object State Name,Object State ID,Constr-ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path		
			Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Object State Name	Object State	2	OST Postcondition Spec	26	calls for the use of
Object State ID	Object State	2	OST Postcondition Spec	26	calls for the use of
Constr-ID	Constraint Set	13	Constraint Set	13	appears as

Used At:	Author: Tom Blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF32	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



Node: E28	Title: Fact Use in Object State Specification	Number:

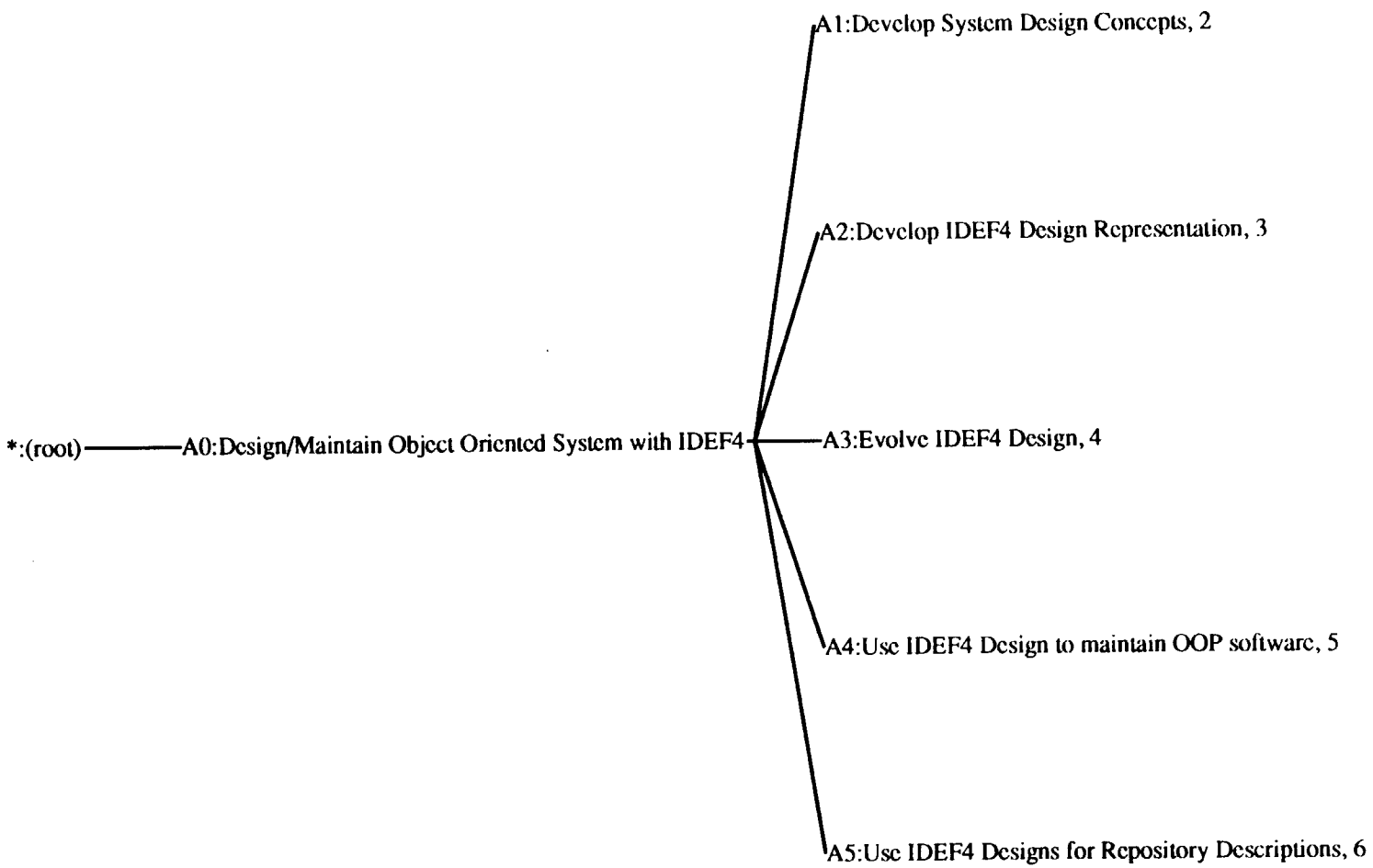
Entity Class Definition: This entity records the use of a fact statement in a description of an object state.

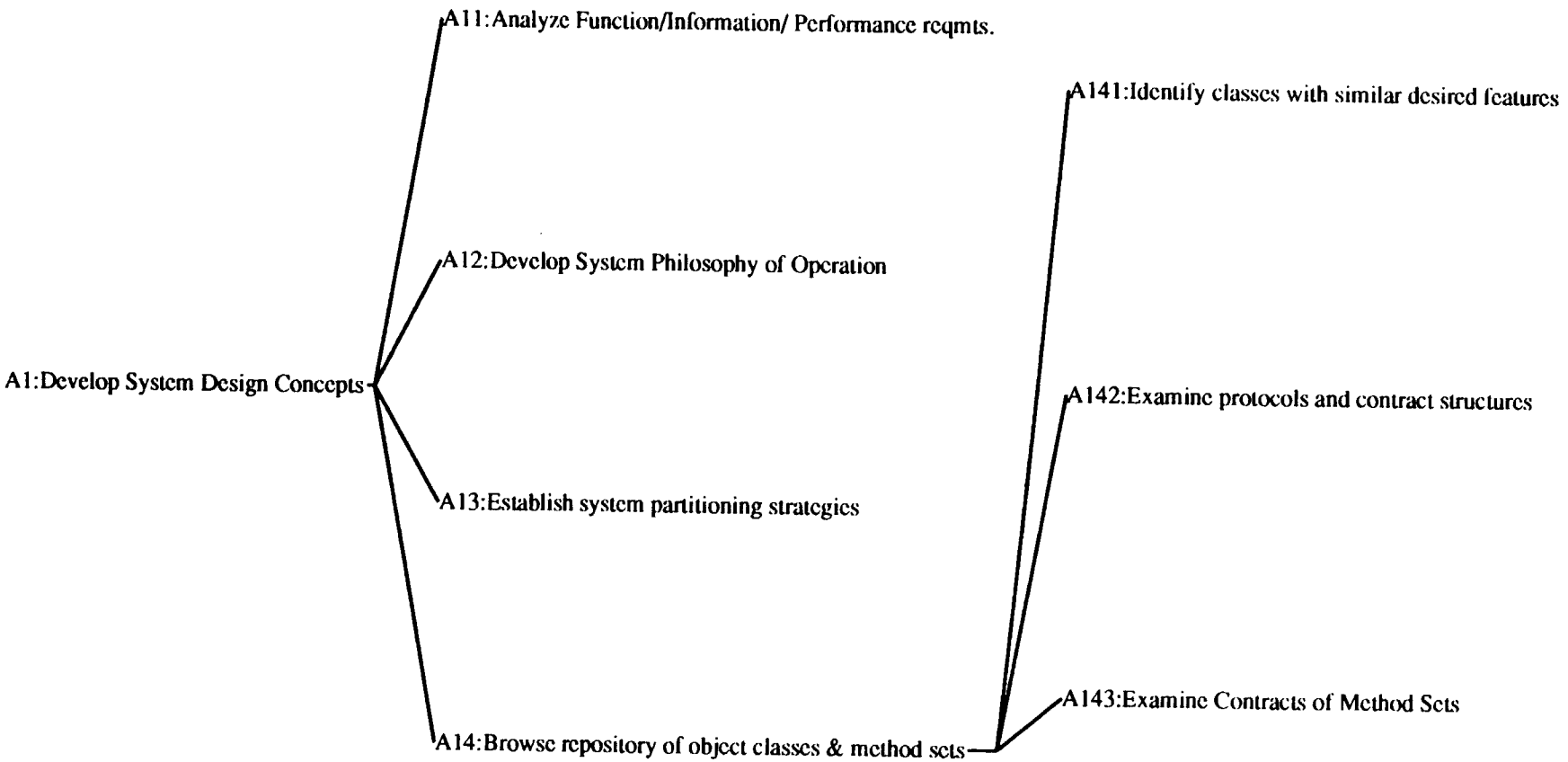
Key Classes:
(Object State ID,Fact Statement ID)

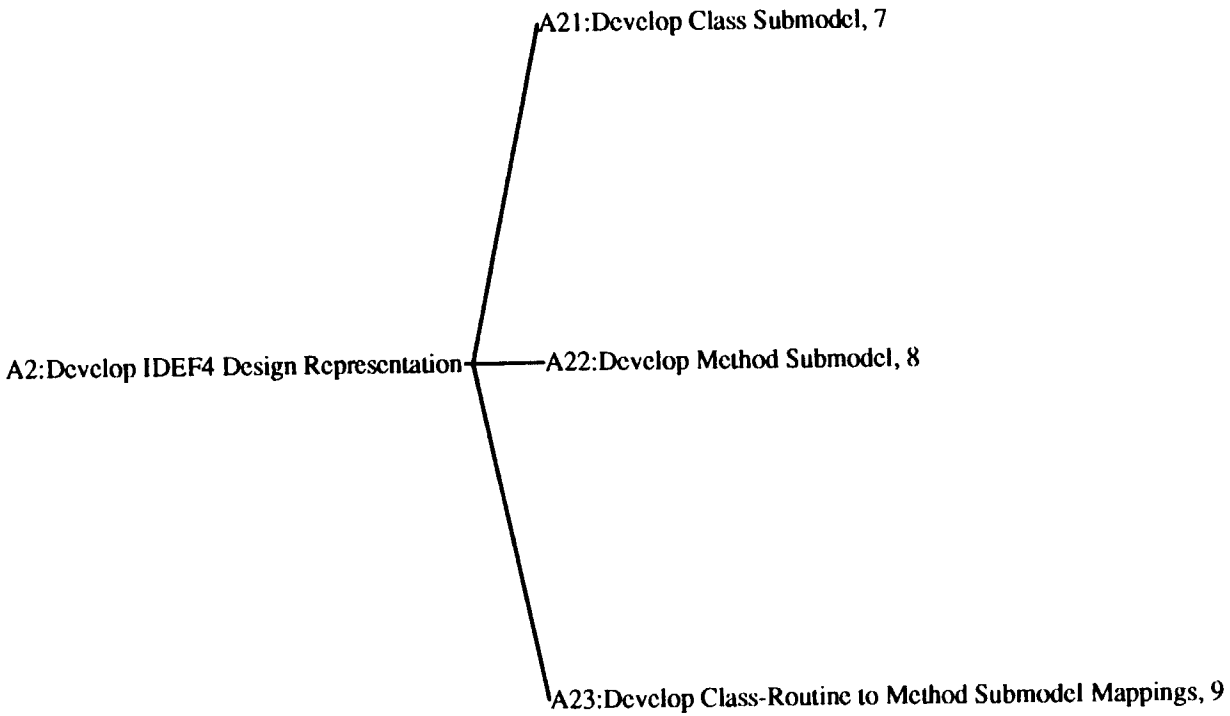
Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Object State Name	Object State	2	Object State	2	is characterized by
Object State ID	Object State	2	Object State	2	is characterized by
Fact Statement ID	Fact Statement	22	Fact Statement	22	may appear in
Fact Relation	Fact Statement	22	Fact Statement	22	may appear in

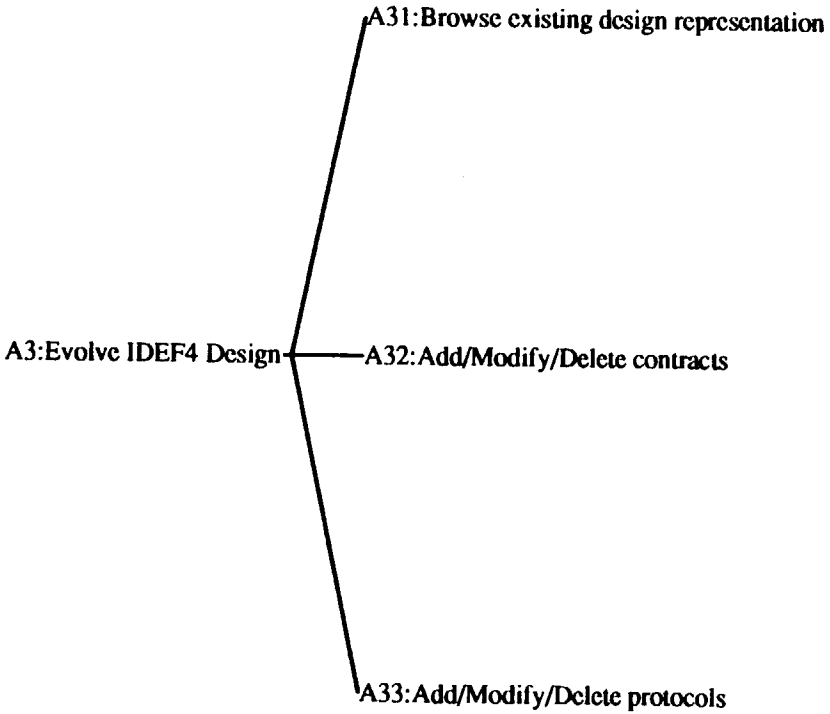
PRECEDING PAGE BLANK NOT FILMED

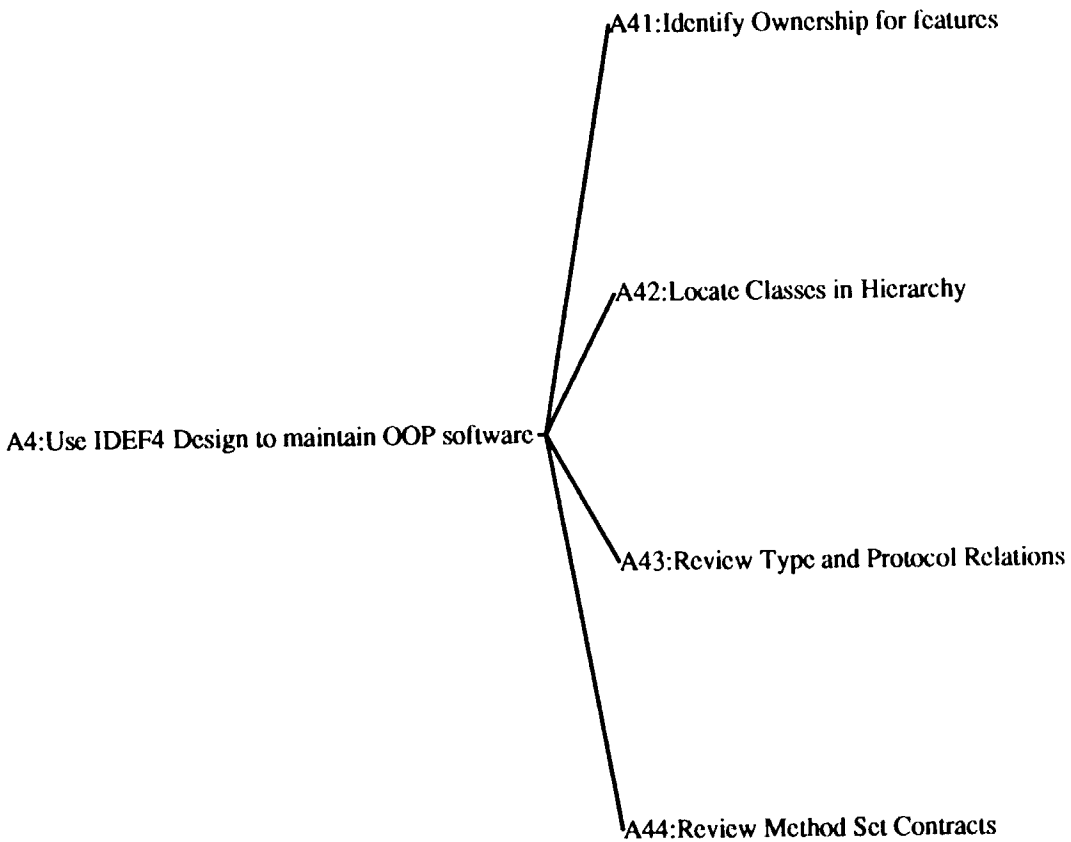
Appendix C: IDEFØ Model of IDEF4

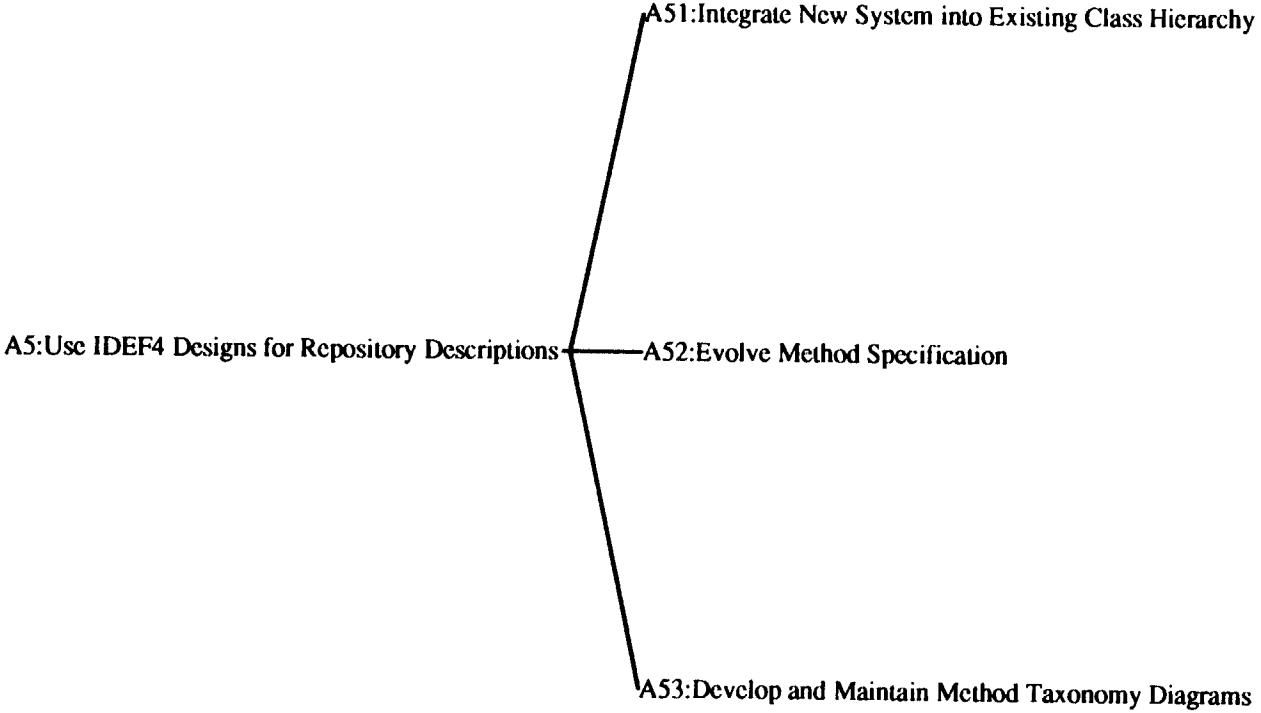

*:(root)
A0:Design/Maintain Object Oriented System with IDEF4
A1:Develop System Design Concepts, 2
A2:Develop IDEF4 Design Representation, 3
A3:Evolve IDEF4 Design, 4
A4:Use IDEF4 Design to maintain OOP software, 5
A5:Use IDEF4 Designs for Repository Descriptions, 6

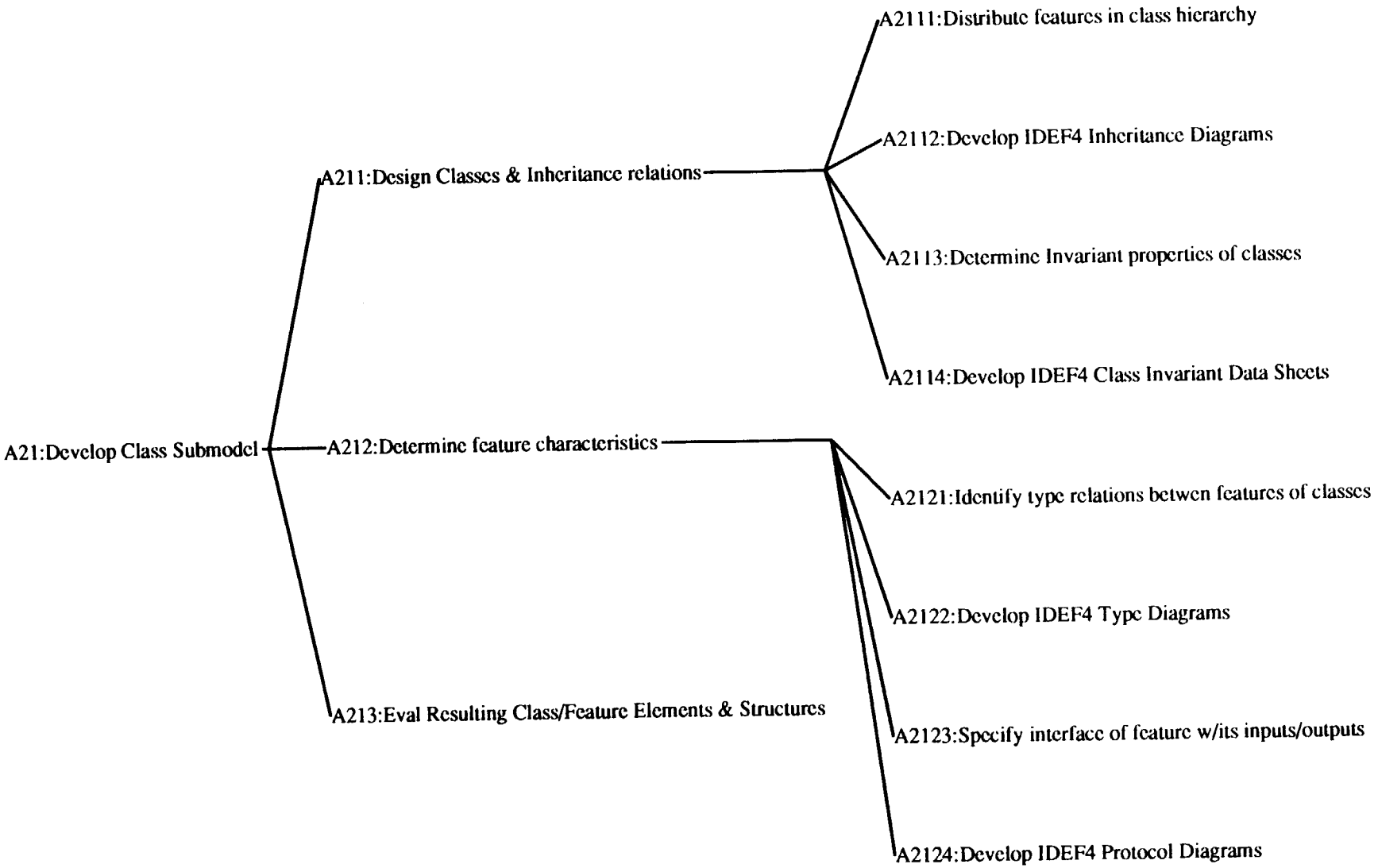

A11:Analyze Function/Information/ Performance reqmts.
A12:Develop System Philosophy of Operation
A1:Develop System Design Concepts
A13:Establish system partitioning strategies
A14:Browse repository of object classes & method sets
A141:Identify classes with similar desired features
A142:Examine protocols and contract structures
A143:Examine Contracts of Method Sets

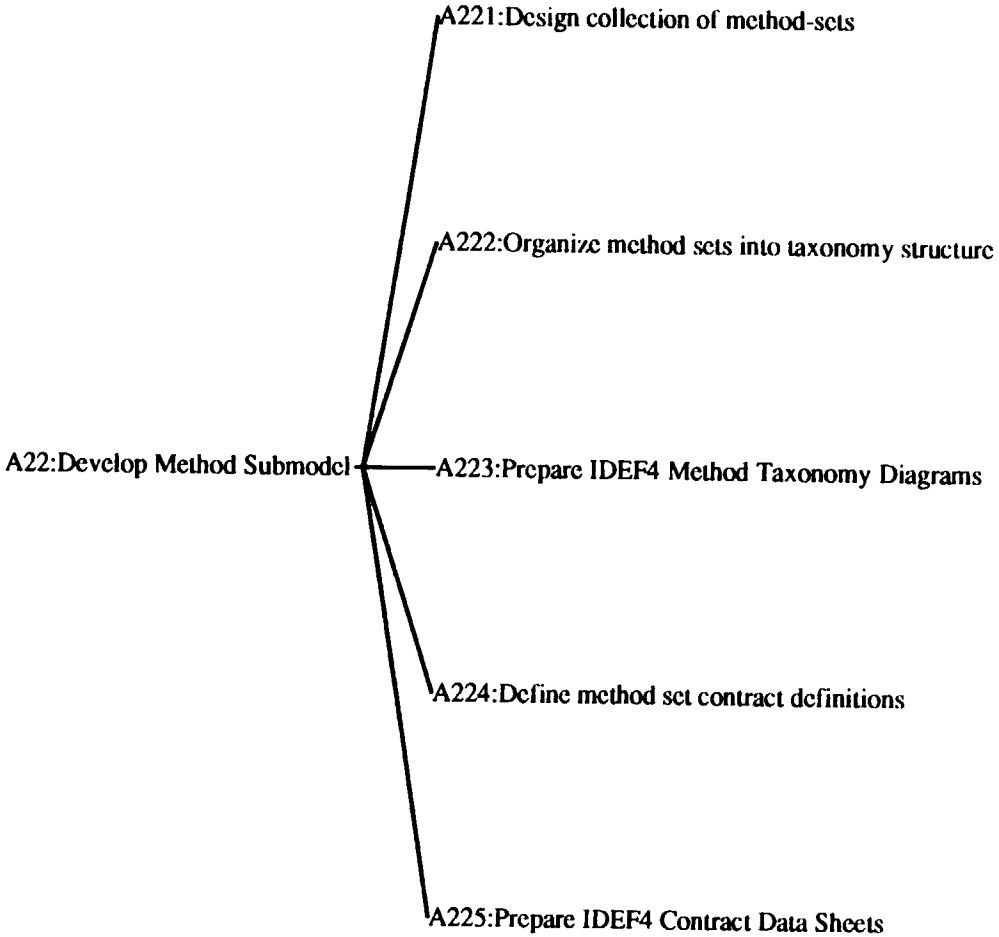

A21:Develop Class Submodel, 7
A2:Develop IDEF4 Design Representation
A22:Develop Method Submodel, 8
A23:Develop Class-Routine to Method Submodel Mappings, 9

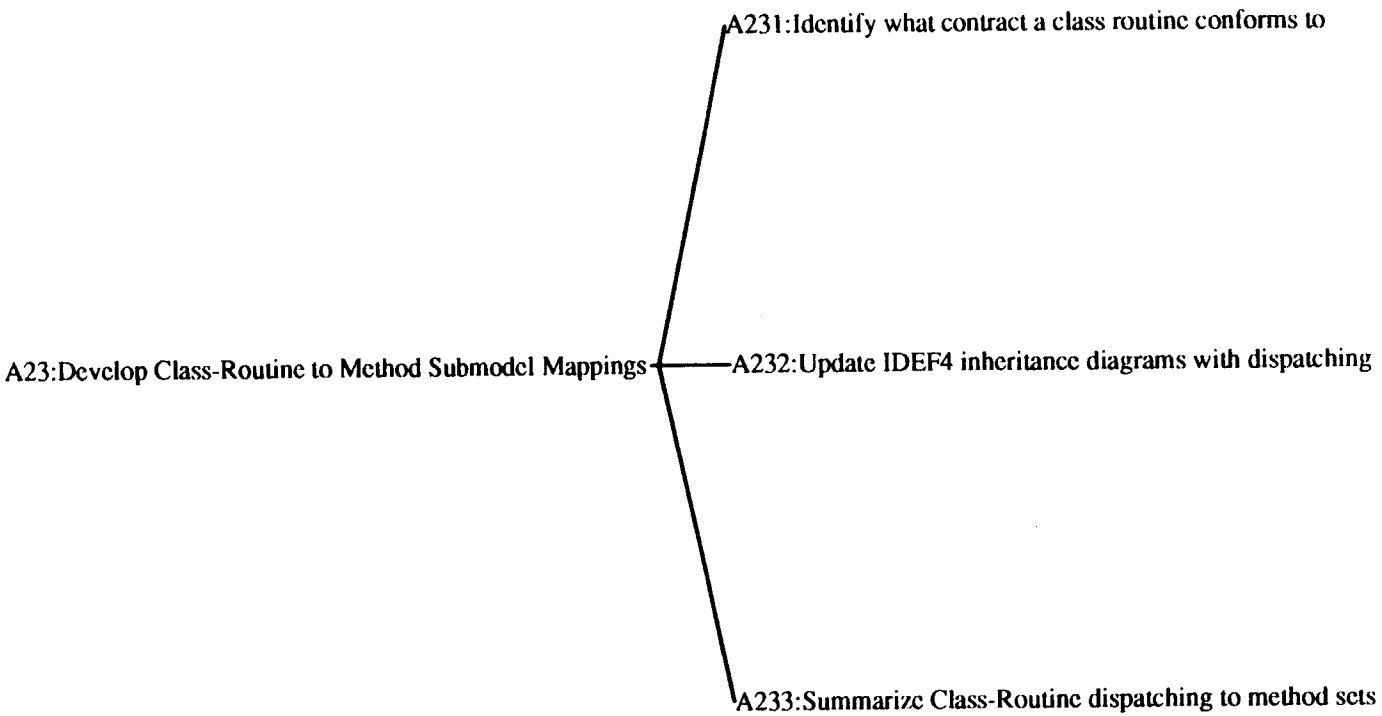

A31:Browse existing design representation
A3:Evolve IDEF4 Design
A32:Add/Modify/Delete contracts
A33:Add/Modify/Delete protocols


A41:Identify Ownership for features
A42:Locate Classes in Hierarchy
A4:Use IDEF4 Design to maintain OOP software
A43:Review Type and Protocol Relations
A44:Review Method Set Contracts


A51:Integrate New System into Existing Class Hierarchy
A5:Use IDEF4 Designs for Repository Descriptions
A52:Evolve Method Specification
A53:Develop and Maintain Method Taxonomy Diagrams


A21:Develop Class Submodel
A211:Design Classes & Inheritance relations
A212:Determine feature characteristics
A213:Eval Resulting Class/Feature Elements & Structures
A2111:Distribute features in class hierarchy
A2112:Develop IDEF4 Inheritance Diagrams
A2113:Determine Invariant properties of classes
A2114:Develop IDEF4 Class Invariant Data Sheets
A2121:Identify type relations betwen features of classes
A2122:Develop IDEF4 Type Diagrams
A2123:Specify interface of feature w/its inputs/outputs
A2124:Develop IDEF4 Protocol Diagrams


A221:Design collection of method-sets
A222:Organize method sets into taxonomy structure
A22:Develop Method Submodel
A223:Prepare IDEF4 Method Taxonomy Diagrams
A224:Define method set contract definitions
A225:Prepare IDEF4 Contract Data Sheets

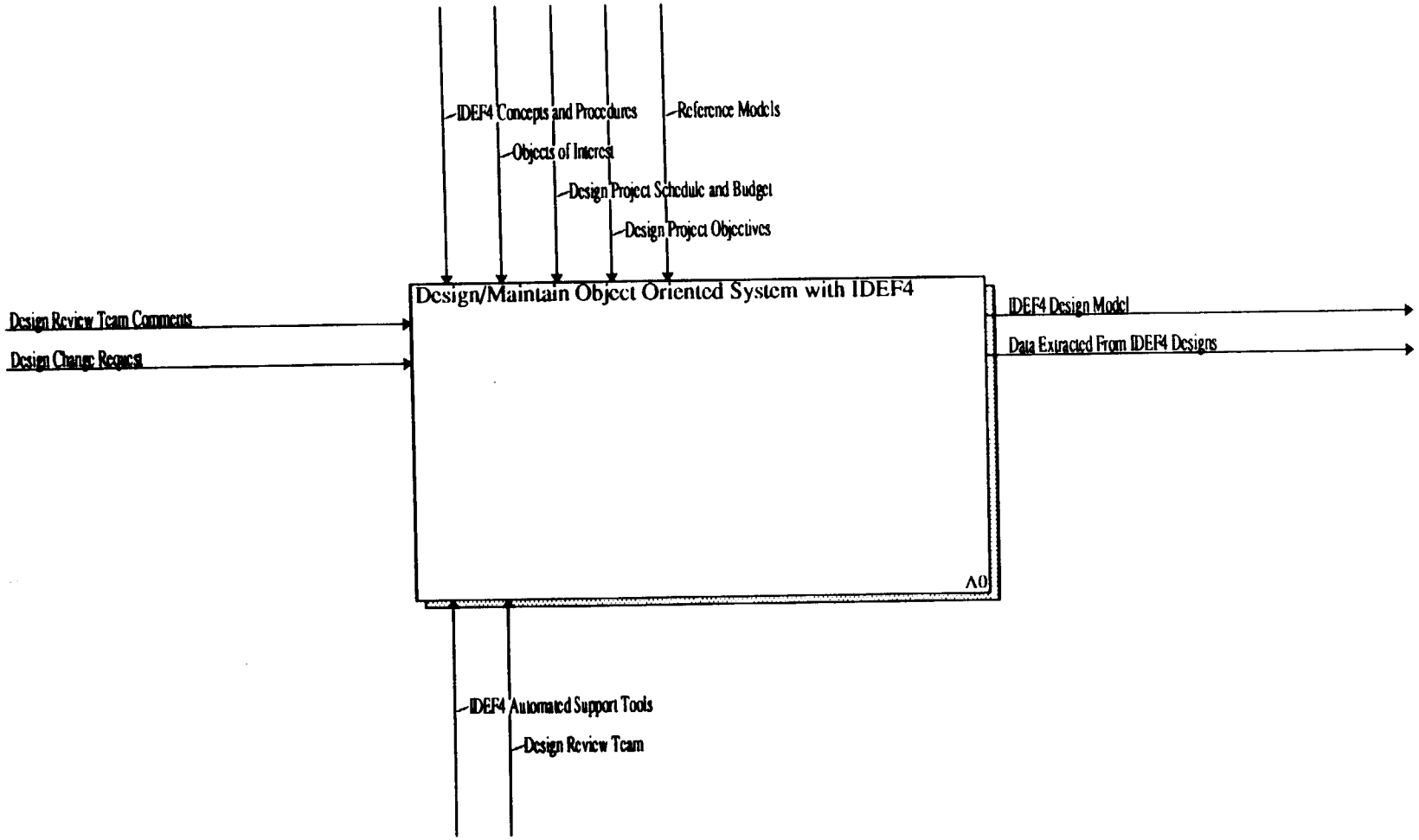

A231:Identify what contract a class routine conforms to
A23:Develop Class-Routine to Method Submodel Mappings
A232:Update IDEF4 inheritance diagrams with dispatching
A233:Summarize Class-Routine dispatching to method sets

Index

DIAGRAMS

INTENTIONALLY BLANK

Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE:	Context
	Project: Idef4req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

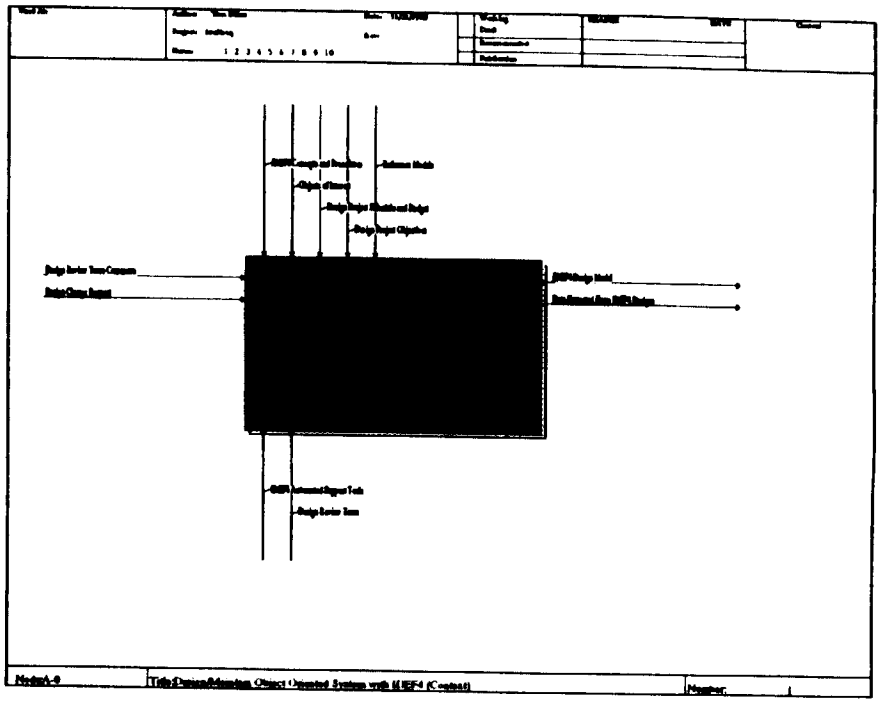


PRECEDING PAGE BLANK NOT FILMED

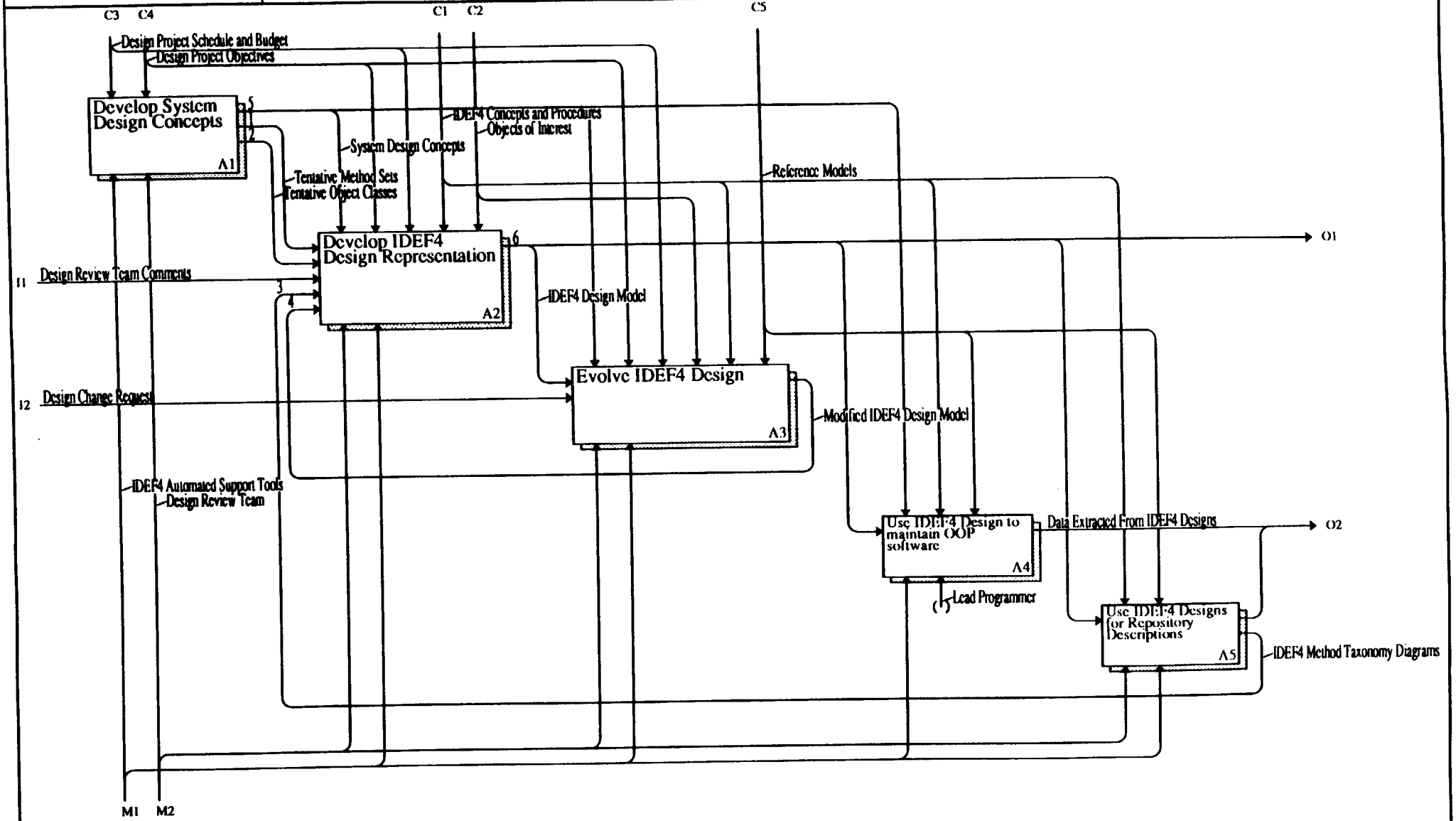
Node:A-0	Title:Design/Maintain Object Oriented System with IDEF4 (Context)	Number: 12

Design/Maintain Object Oriented System with IDEF4

IDEF4 provides the ability to design and maintain an object oriented software system. It allows for the development of extensive designs as well as the ability to examine the design for information that might assist in the maintenance of the implemented software system.



Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef4req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



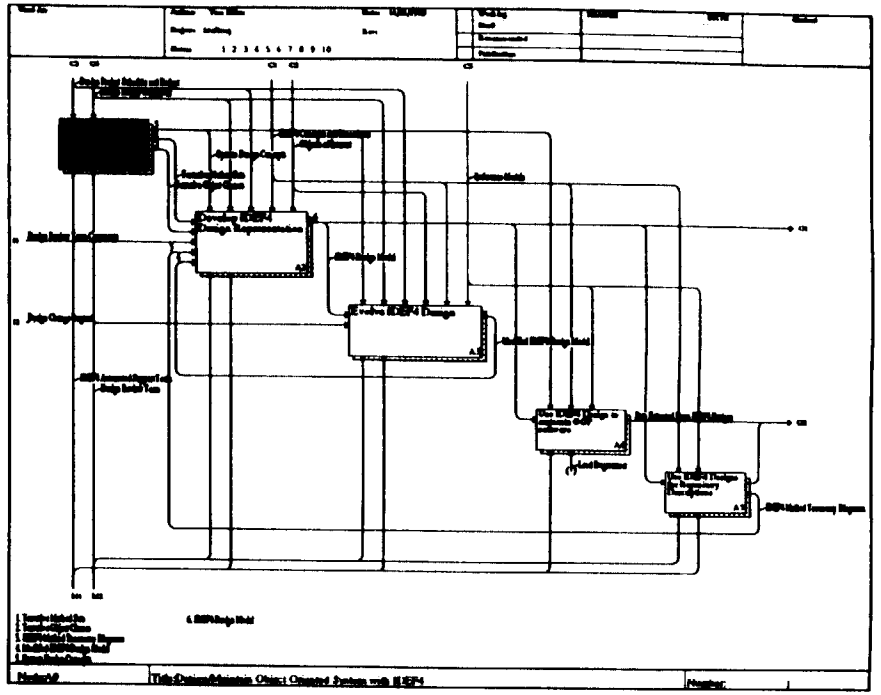
1. Tentative Method Sets
2. Tentative Object Classes
3. IDEF4 Method Taxonomy Diagrams
4. Modified IDEF4 Design Model
5. System Design Concepts

6. IDEF4 Design Model

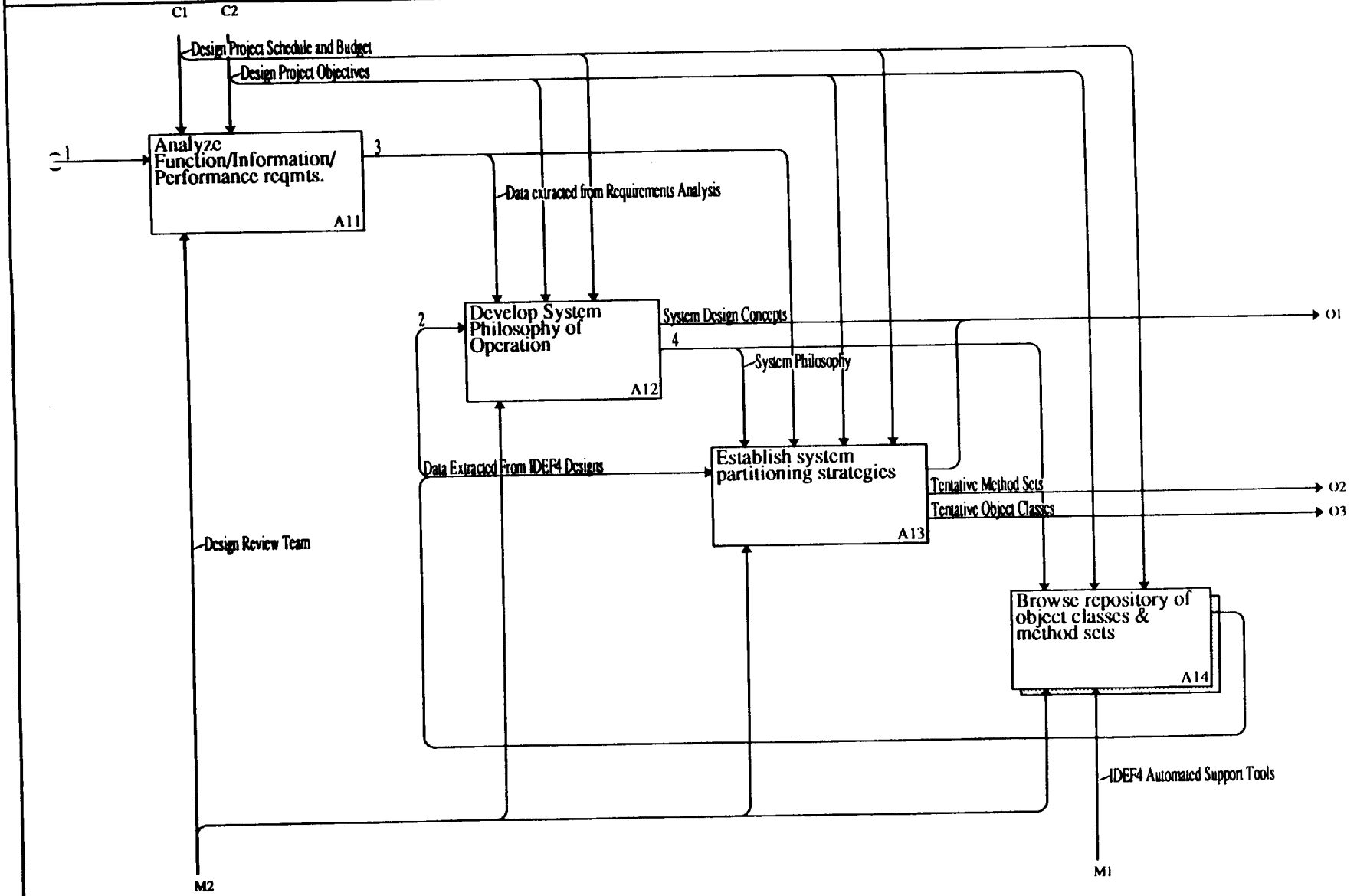
Node:A0	Title:Design/Maintain Object Oriented System with IDEF4	Number: 14

Develop System Design Concepts

At this point, the design team is attempting to define the "look and feel" of the system. The output of this activity is a list of general goals that the tool should attempt to meet in fulfilling its functional requirements.



Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef4req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

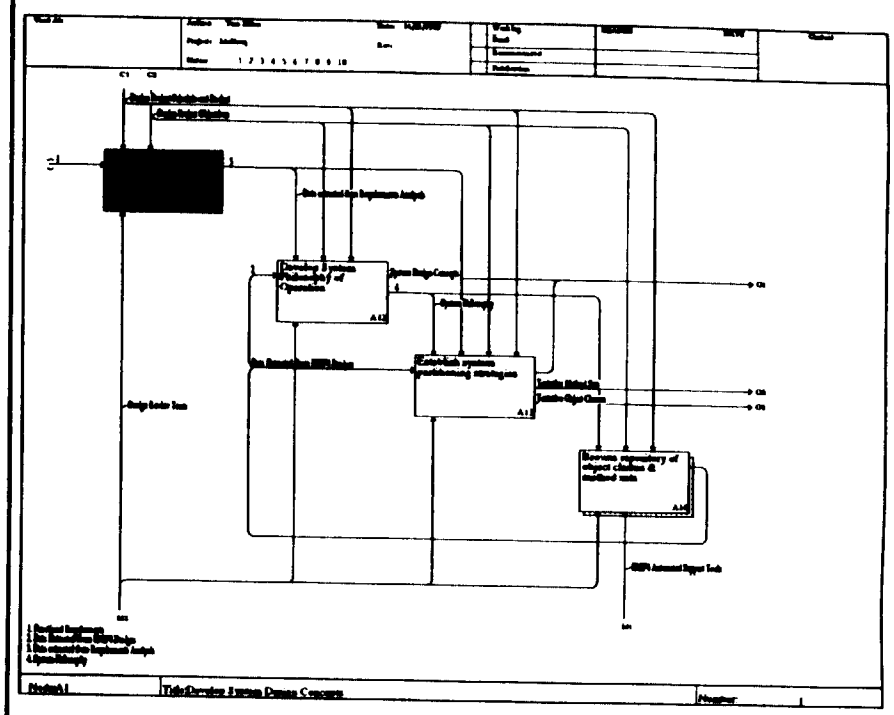


1. Functional Requirements
2. Data Extracted From IDEF4 Designs
3. Data extracted from Requirements Analysis
4. System Philosophy

Node:A1	Title:Develop System Design Concepts	Number: 16

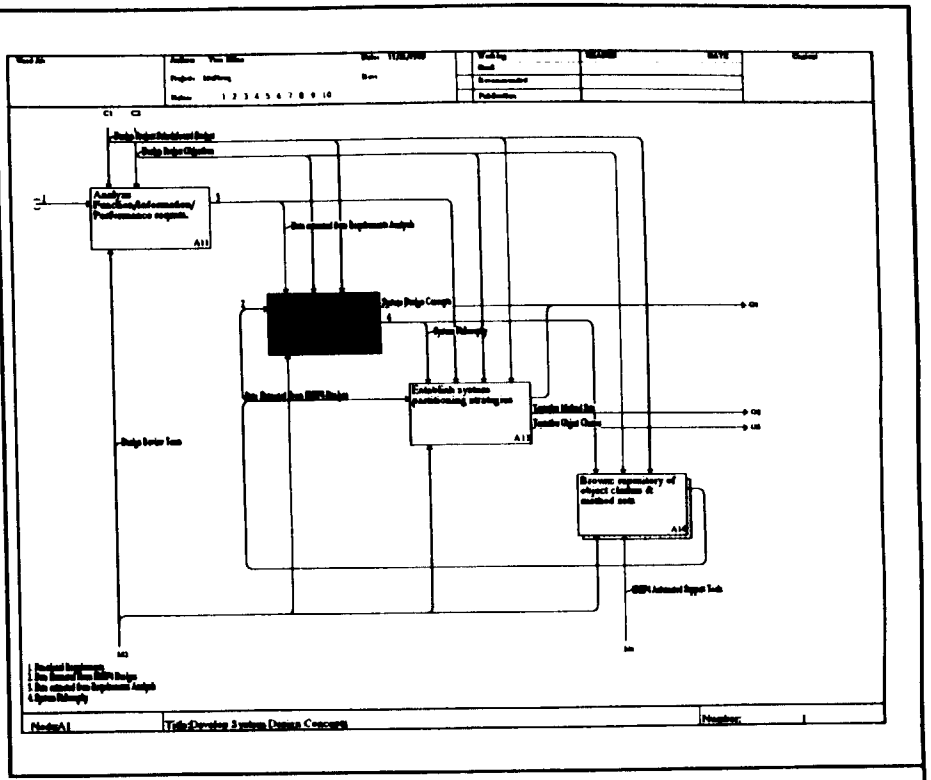
Analyze Function/Information/ Performance reqmts.

Here, the functional requirements are examined to determine the amount of freedom will be allowed in the design of the system.



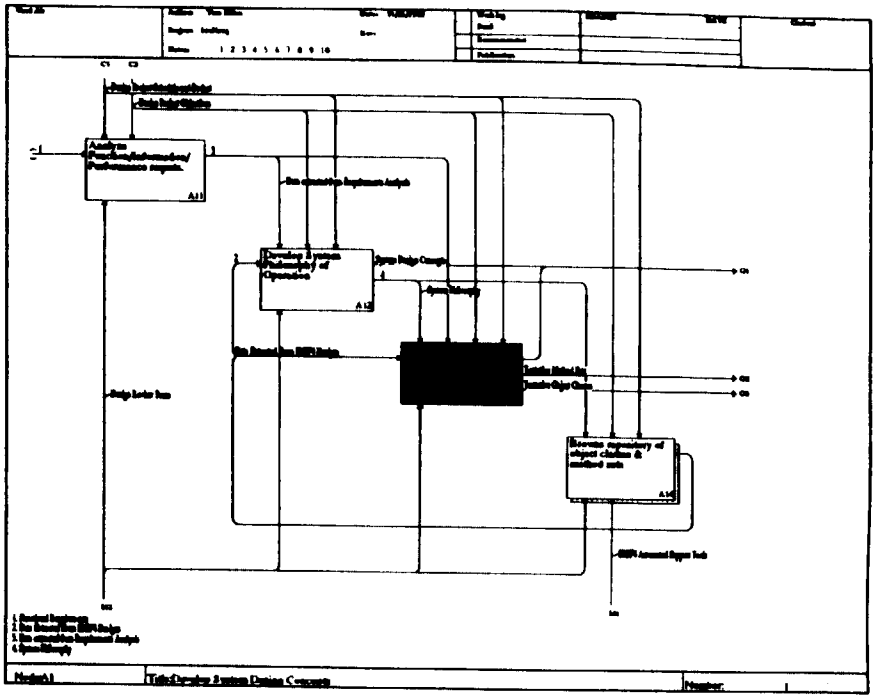
Develop System Philosophy of Operation

This is the actual definition of the "look and feel" of the system.



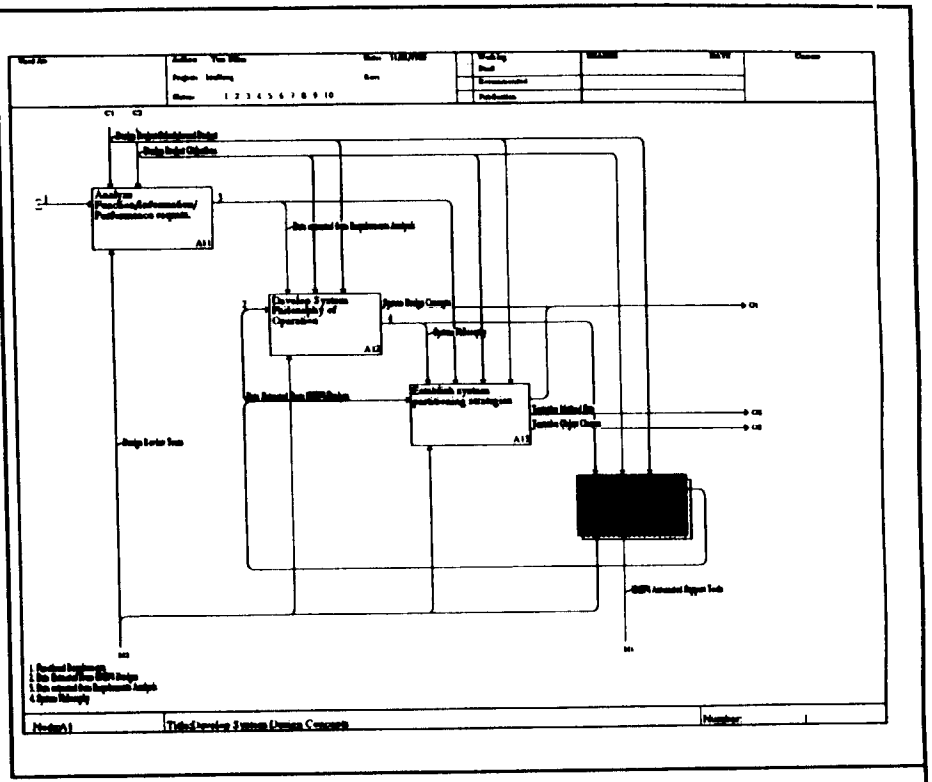
Establish system partitioning strategies

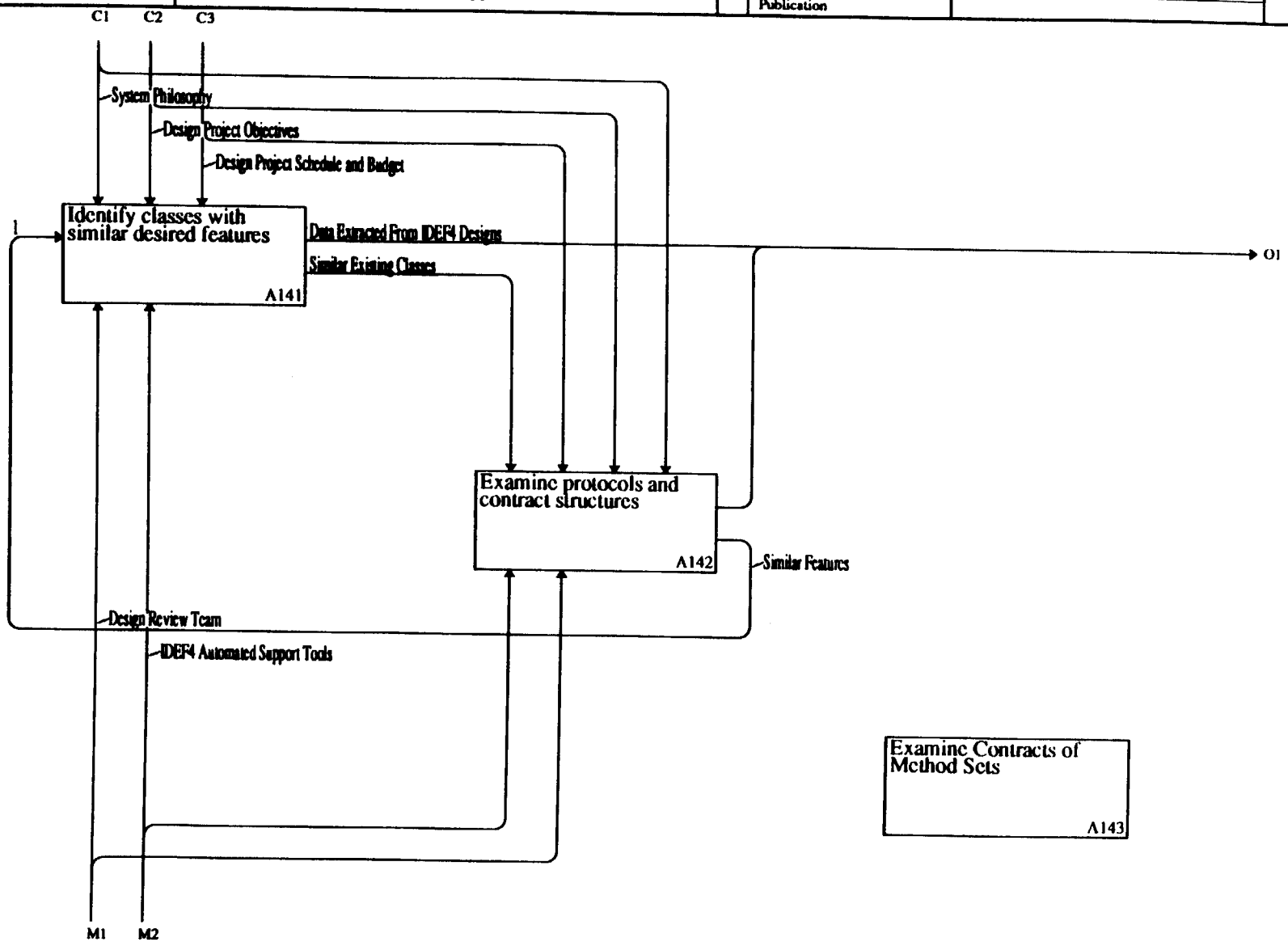
This establishes an initial attempt at subdividing the system into logical units.



Browse repository of object classes & method sets

At this point, examination of other systems occurs to determine what existing modules could be used as is or with slight modification in the new system.

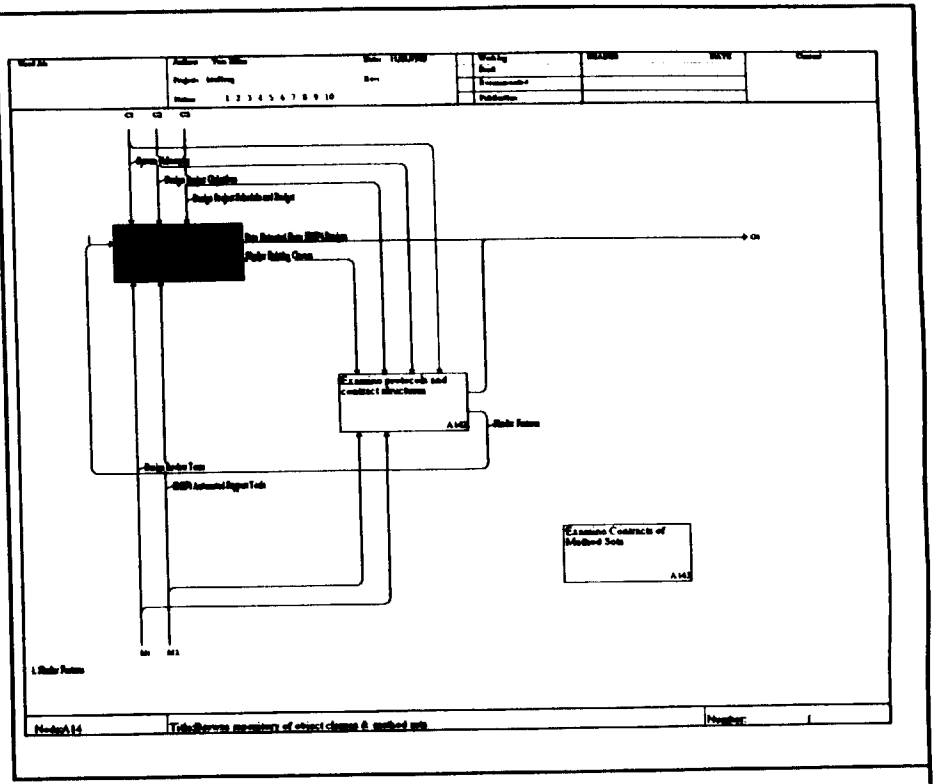




1. Similar Features

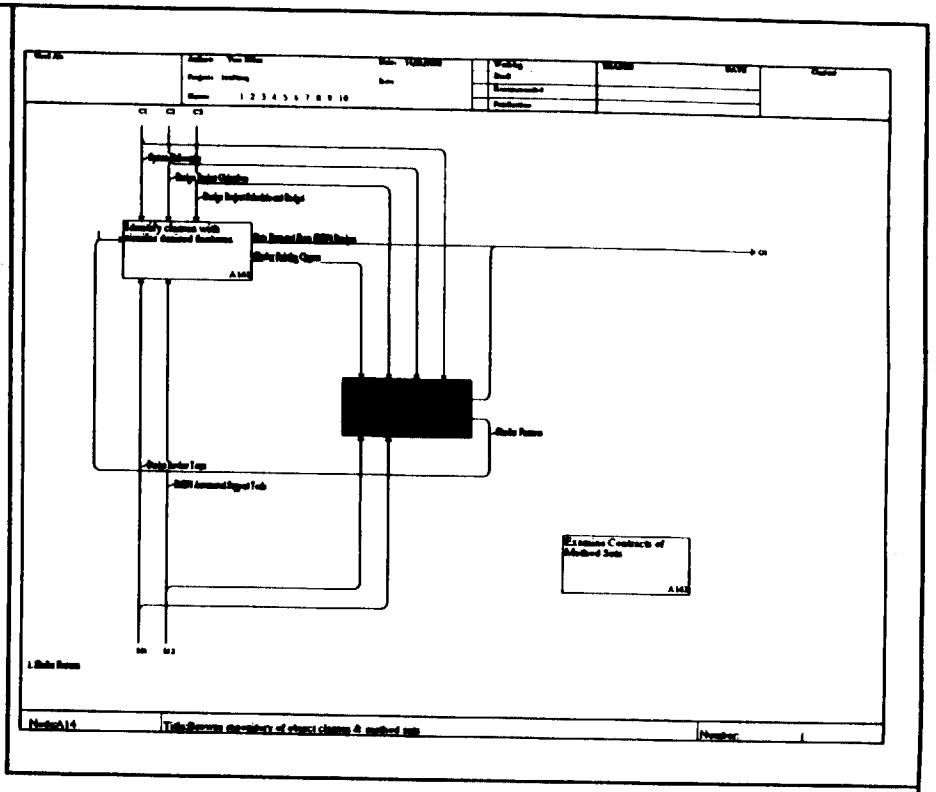
Identify classes with similar desired features

Here, the design team tries to identify classes that provide the same functionality that is required in the new system.

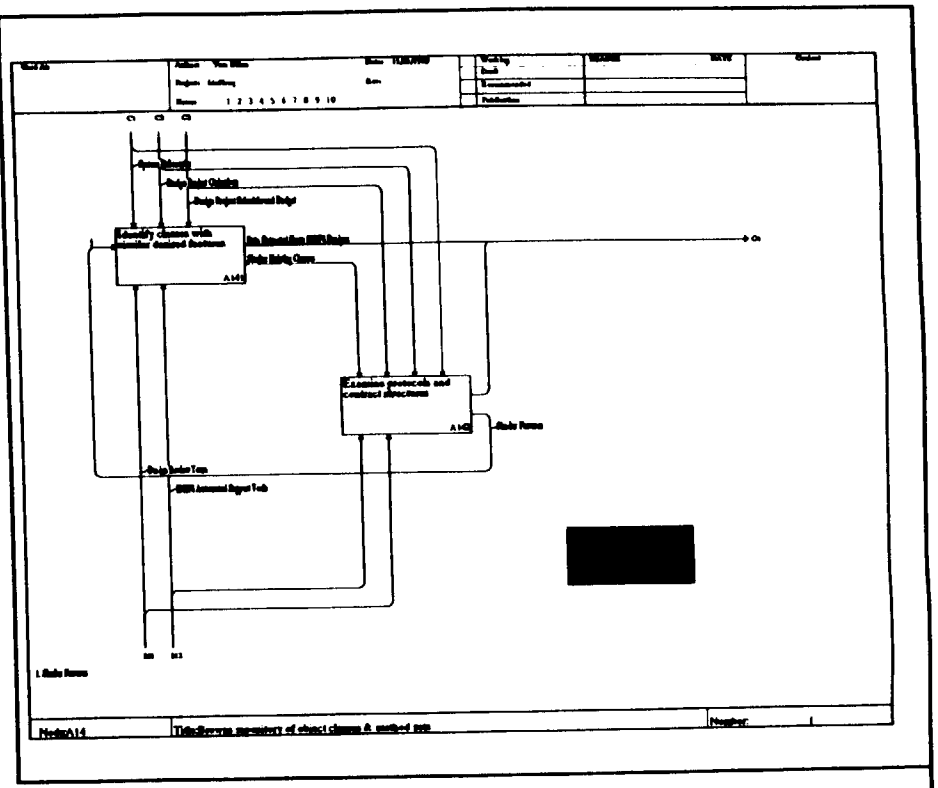


Examine protocols and contract structures

After identifying classes that could potentially be used in the new system, the classes are examined more closely to see how well they match the functional requirements of the new system.



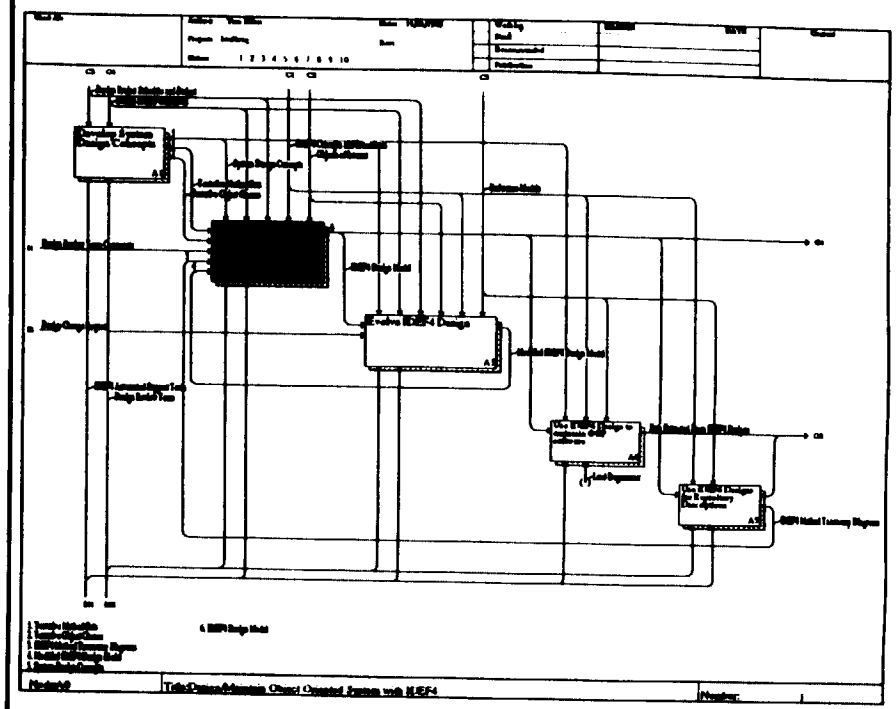
Examine Contracts of Method Sets



--This space intentionally left blank--

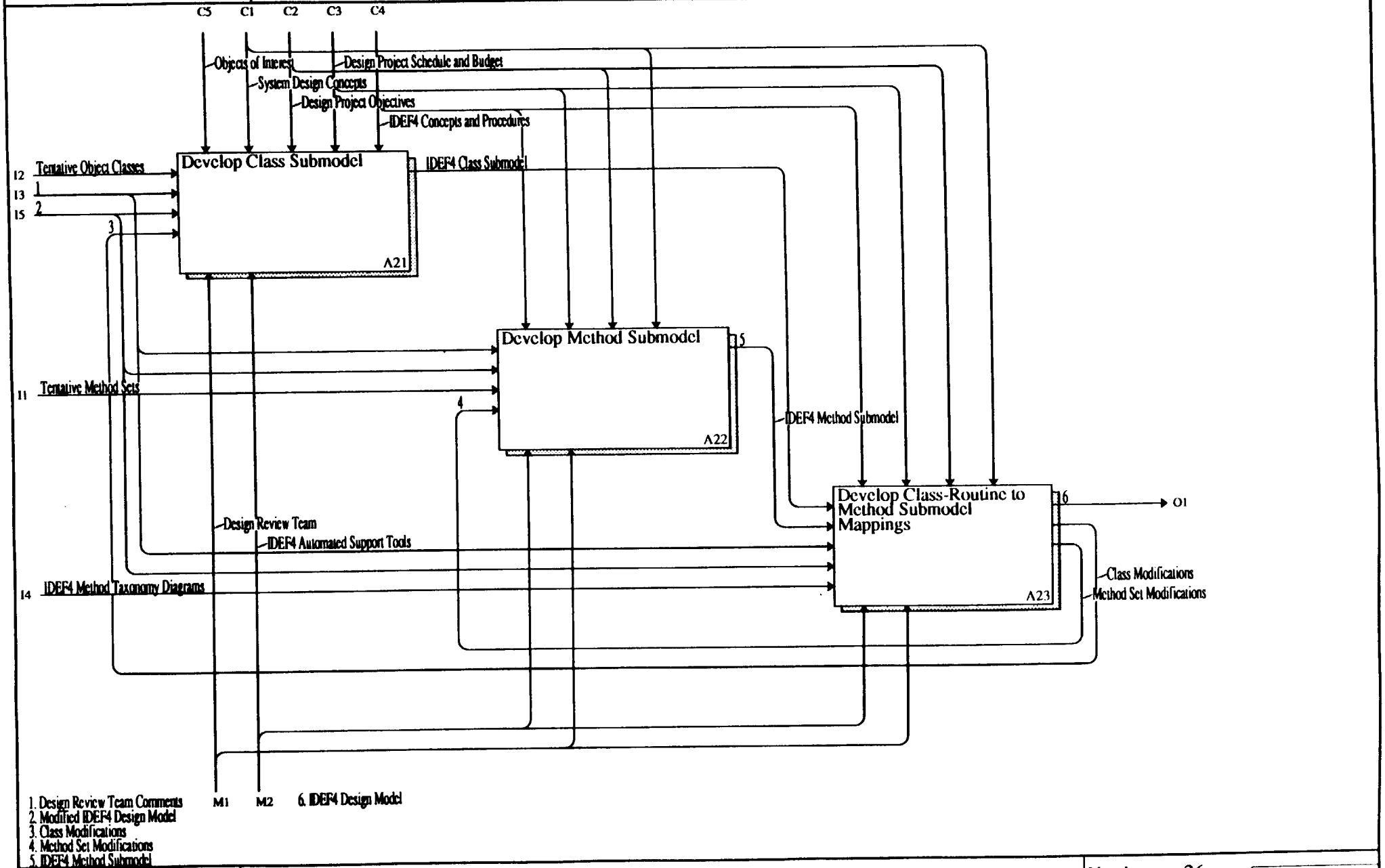
Develop IDEF4 Design Representation

Here is where the actual design work occurs. The necessary classes and there required features are defined along with the method sets that define the functionality that be satisfied by that set.



ORIGINAL PAGE IS
OF POOR QUALITY

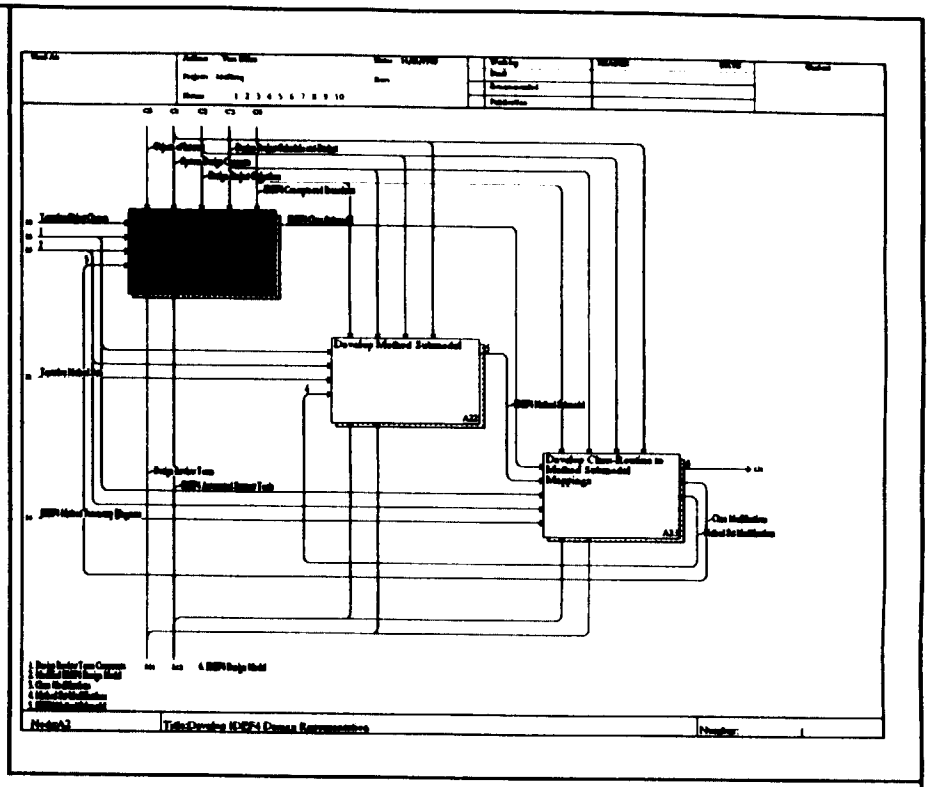
Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef4req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



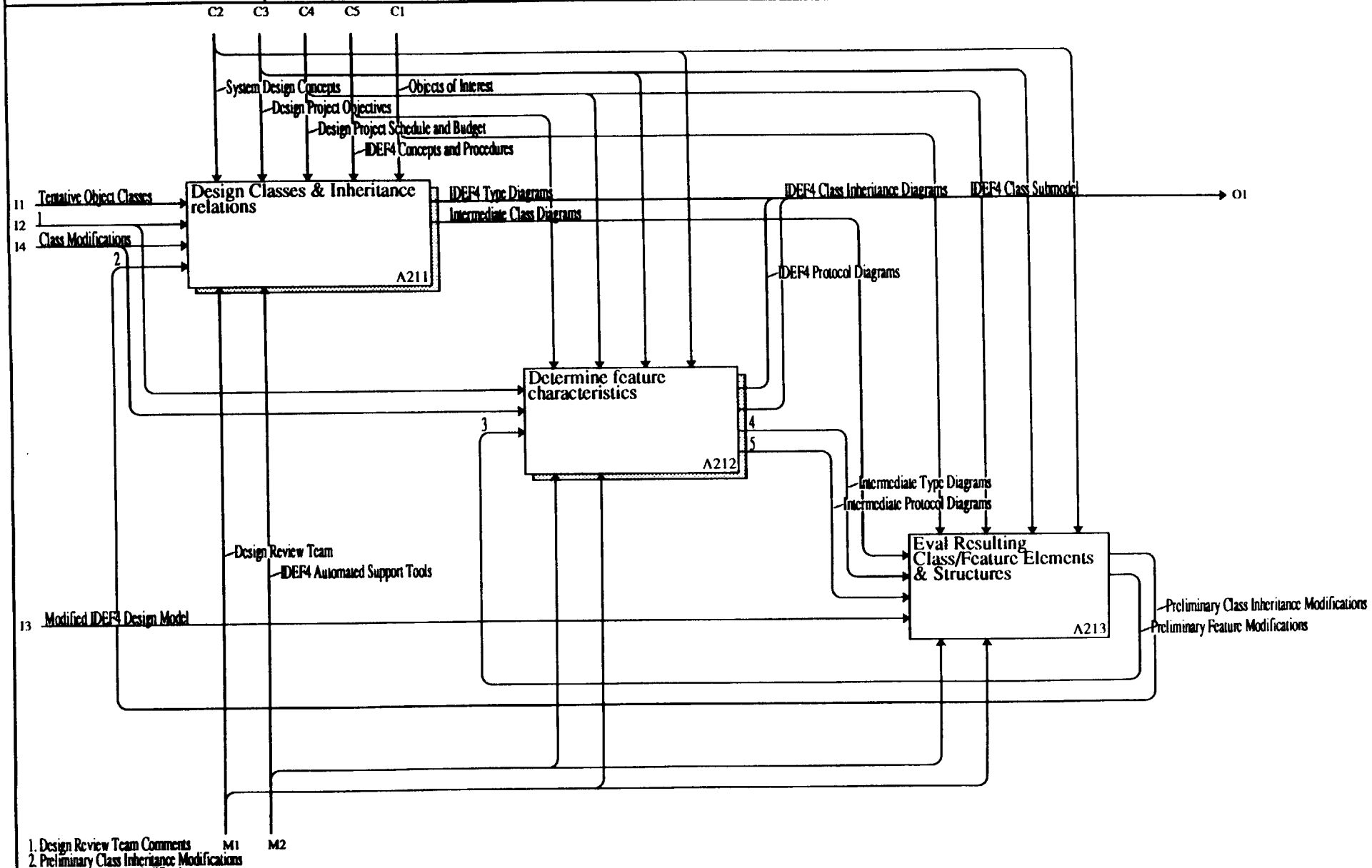
Node:A2	Title:Develop IDEF4 Design Representation	Number: 26

Develop Class Submodel

This entails the process of defining the classes and their inheritance relationships, assigning features to classes and assigning the feature types. Additionaly, the protocols for routine features are specified here.



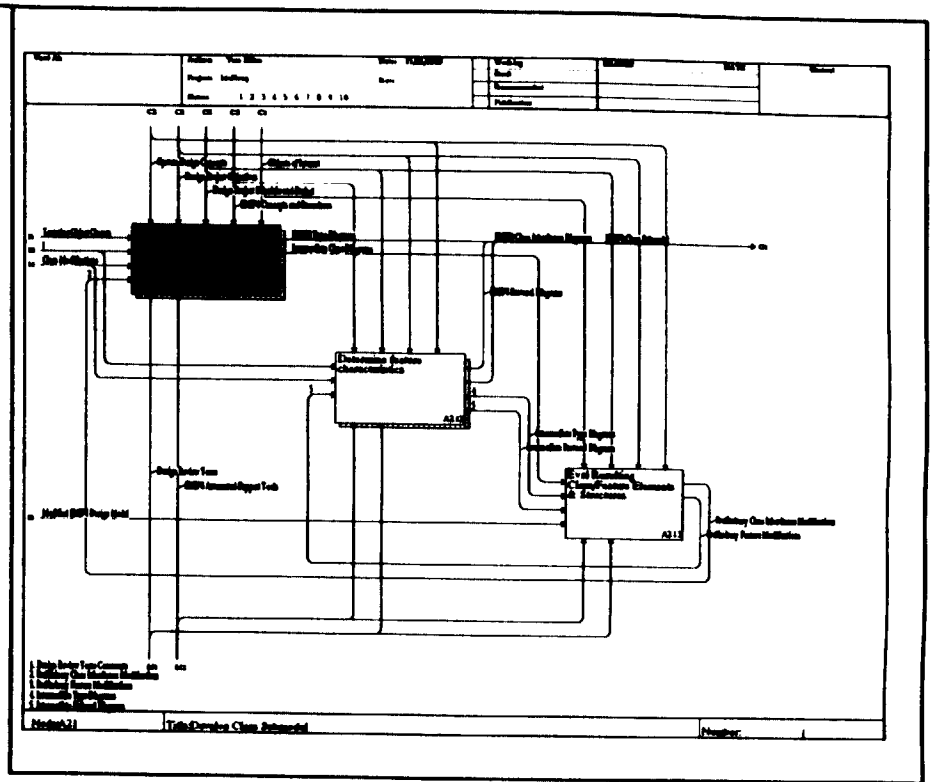
Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef4req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



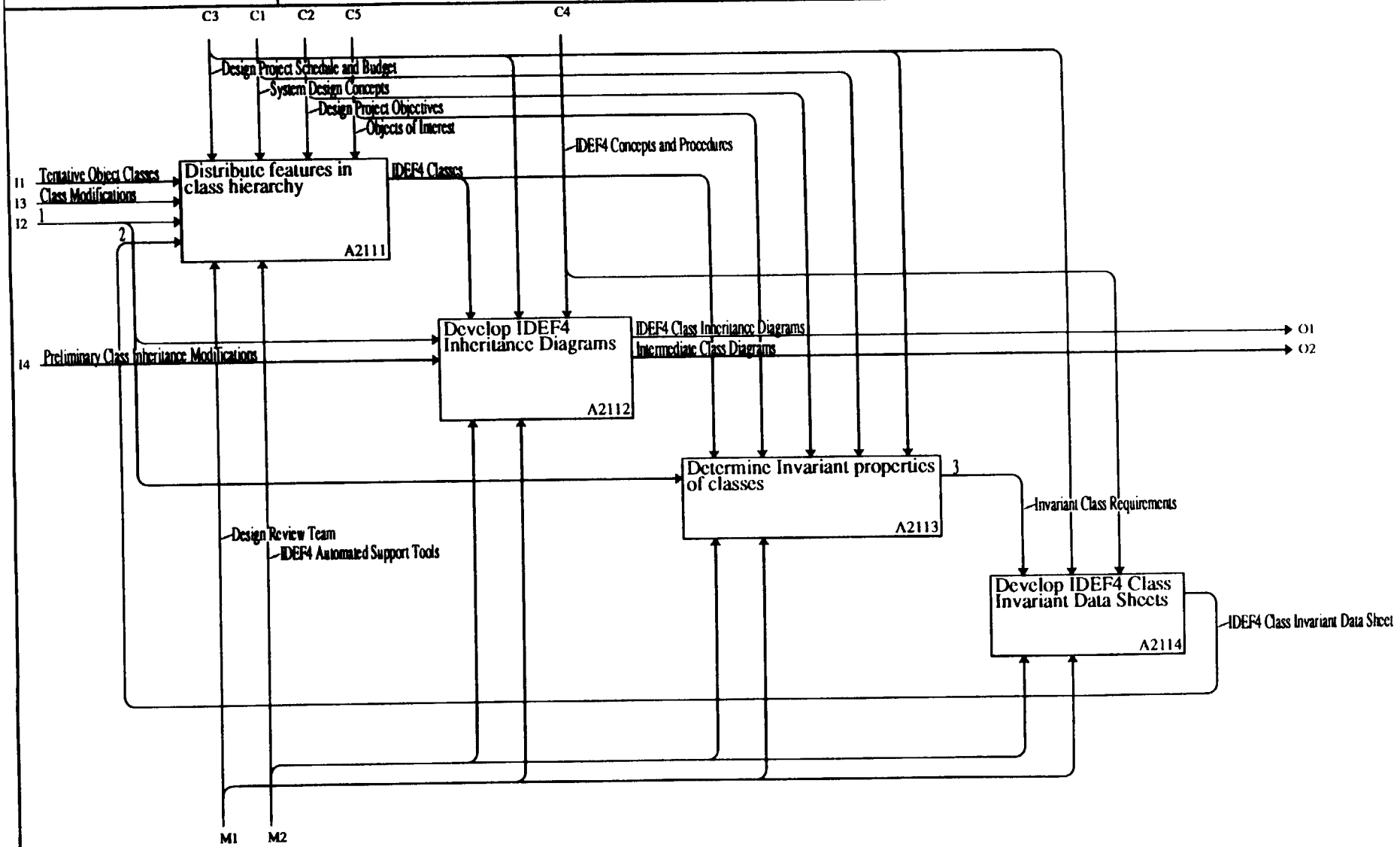
Node:A21	Title:Develop Class Submodel	Number: 28

Design Classes & Inheritance relations

Within this activity, the various classes that will make up the system design are defined, along with the inheritance relations between classes.



Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef4req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

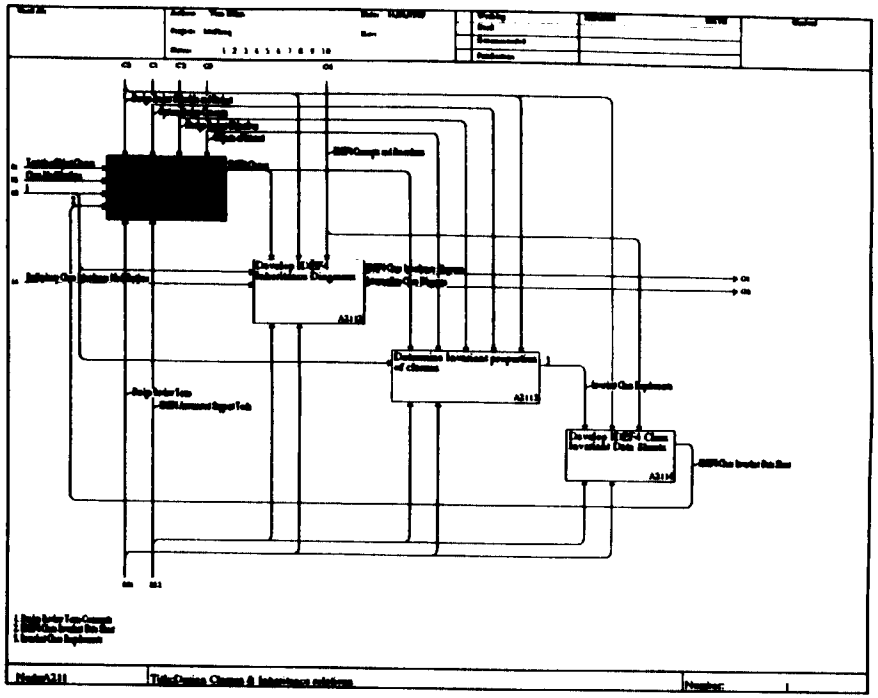


1. Design Review Team Comments
2. IDEF4 Class Invariant Data Sheet
3. Invariant Class Requirements

Node:A211	Title:Design Classes & Inheritance relations	Number: 30

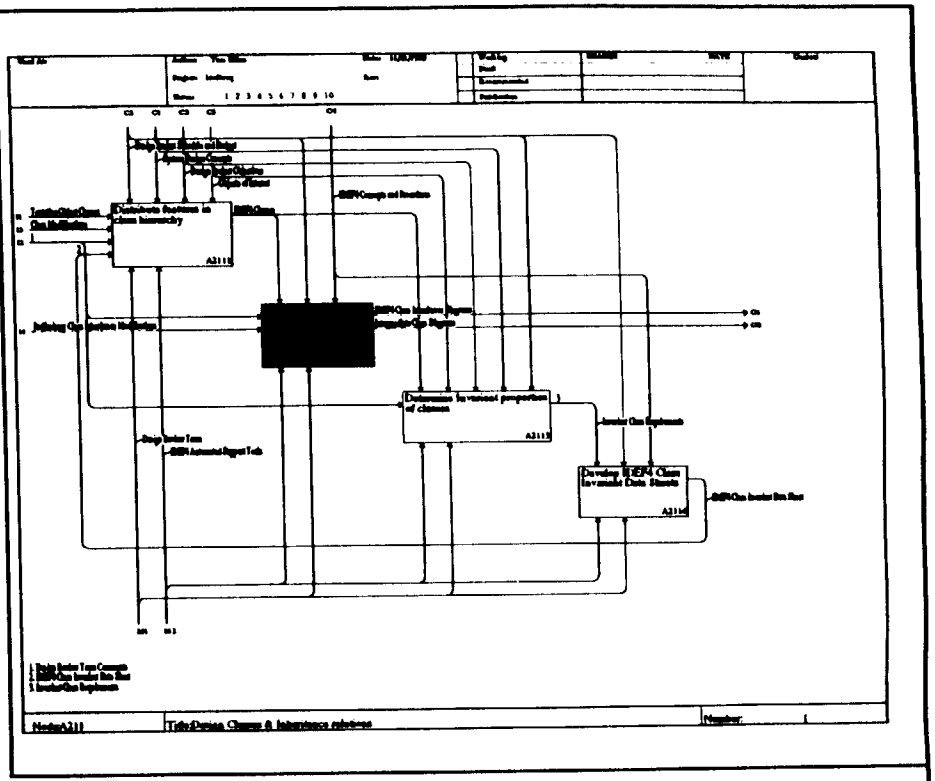
Distribute features in class hierarchy

At this point, features are associated with their respective classes. The features define the character of the class. Also, it is at this point that the role of the feature is specified (i.e., routine, attribute, slot, function, or procedure)



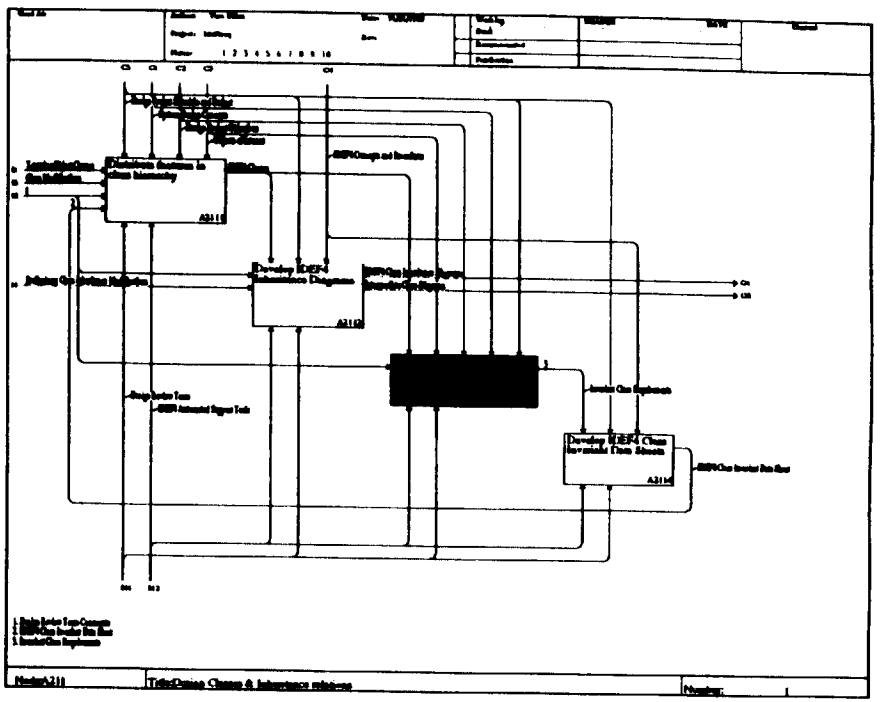
Develop IDEF4 Inheritance Diagrams

Because of the size of the class submodel, it is impratical to display the entire diagram. Instead, several smaller diagrams, or views of the inheritance relationships are defined. Each Inheritance Diagram specified by the modeler has a set of classes whose inheritance relationships are to be displayed in the diagram.



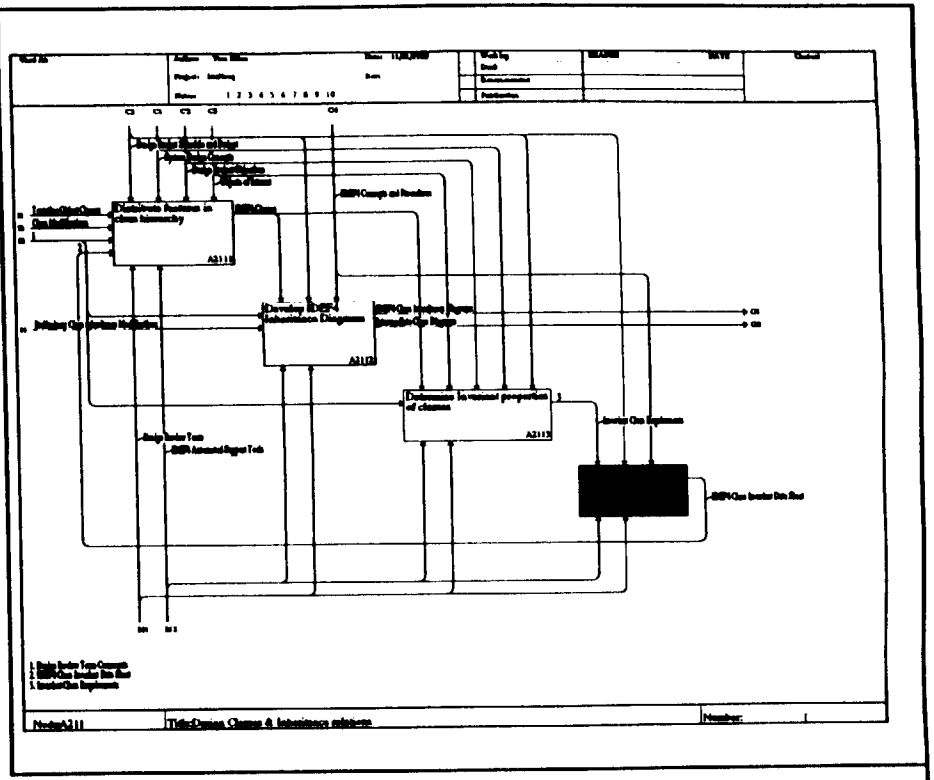
Determine Invariant properties of classes

An invariant is a constraint on class instances that must be satisfied for all instances in the class at all times. This activity closely examines the classes to determine what invariants should be defined for that class.



Develop IDEF4 Class Invariant Data Sheets

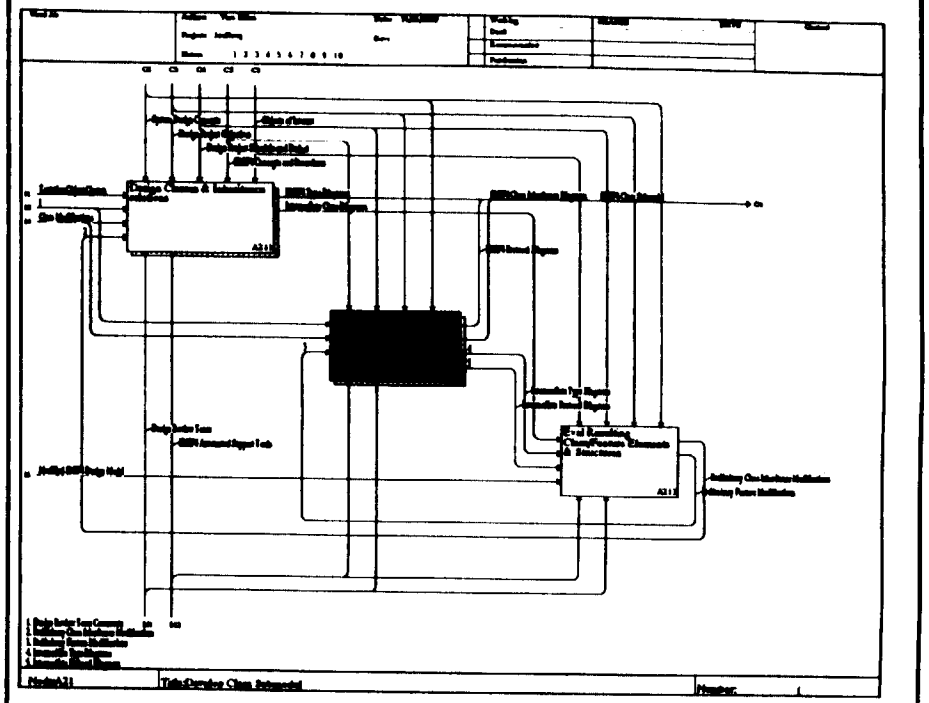
The invariant constraints that have been identified for a class are included in the class description by giving a textual description of the invariants in a class invariant data sheet associated with every class.



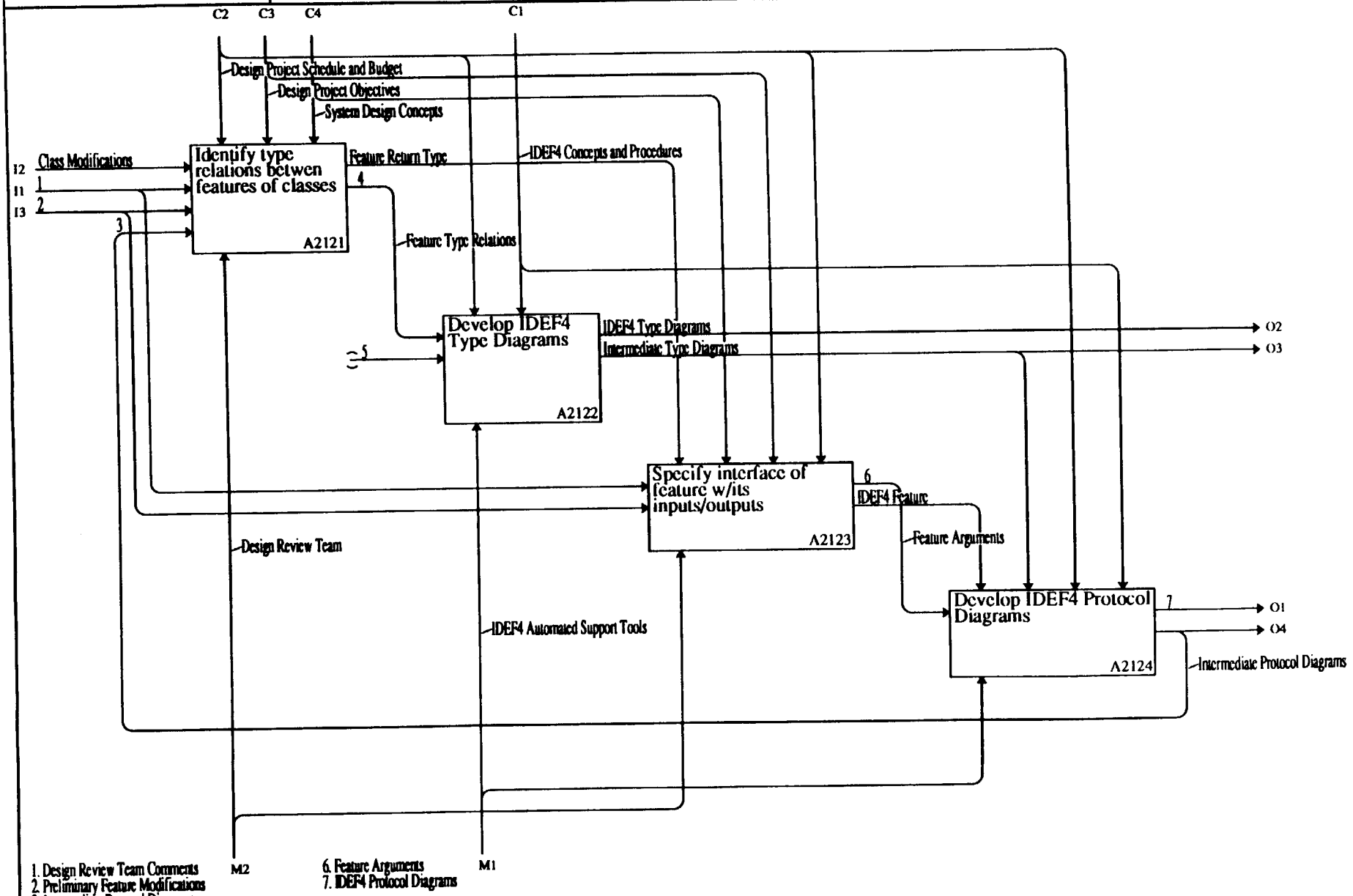
ORIGINAL PAGE IS
OF POOR QUALITY

Determine feature characteristics

This activity is responsible for defining the behavior of the features of the various classes. It involves the specification of the types that a feature may take or return and defining the protocol of arguments to routine features.



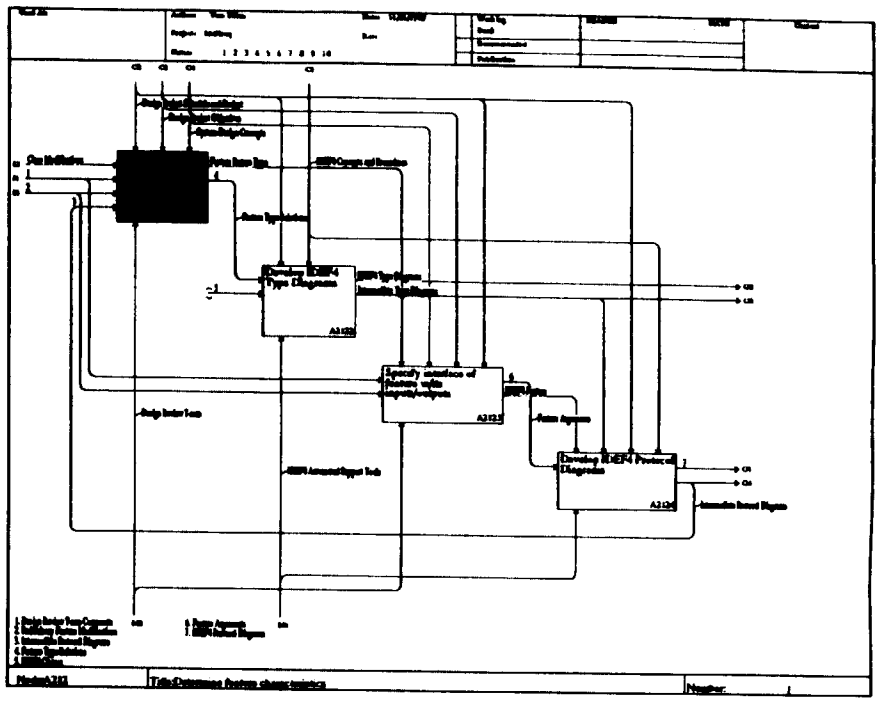
Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef4req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



Node:A212	Title:Determine feature characteristics	Number: 36

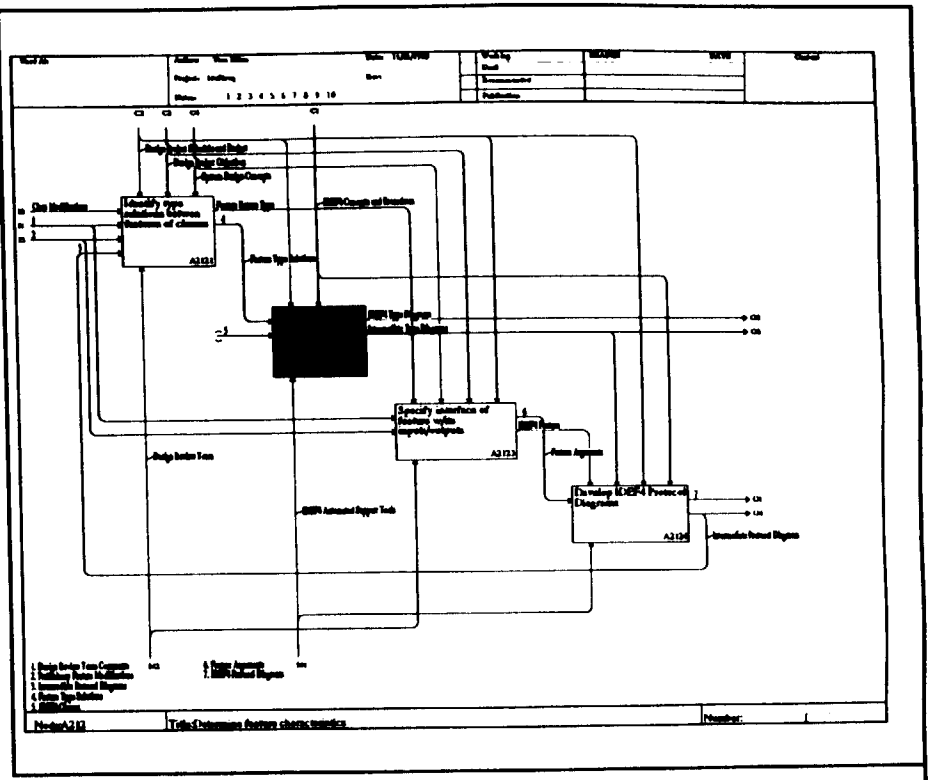
Identify type relations betwen features of classes

Indicate the class types that attribute features may take or routine features may return.



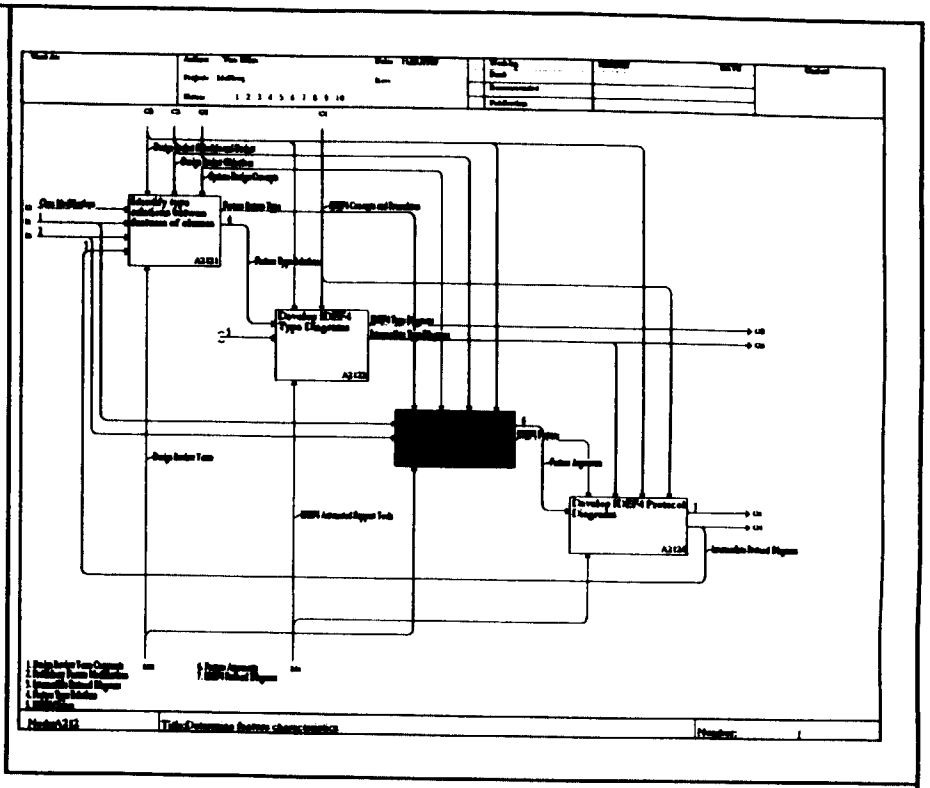
Develop IDEF4 Type Diagrams

This is very similar to defining the Class Inheritance Diagrams since, again, the size of the class submodel restricts the number of classes that can be reasonably displayed. As a result, the user indicates a set of classes to include in the various Type Diagrams. The relationship links are then indicated on the diagram. Two different presentations of the type information are supported, but most use the physical link instead of the graphical link since it is more intuitively ojbious



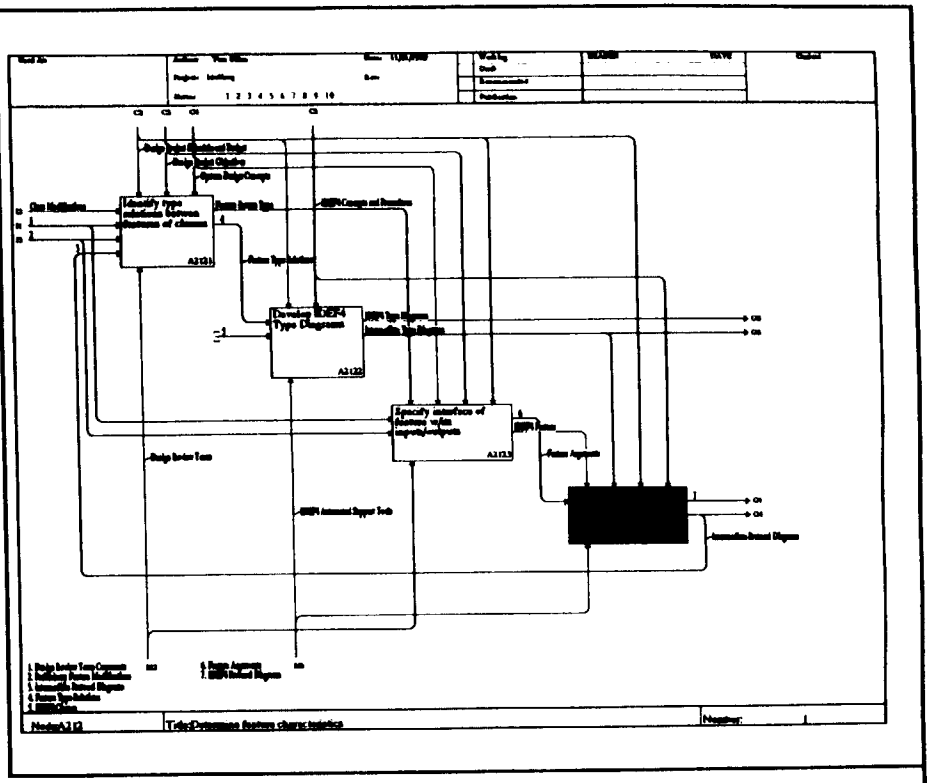
Specify interface of feature w/its inputs/outputs

A routine feature can require that arguments be supplied before that routines value can be determined. It is helpful to represent the types of the arguments that must be supplied to the routine. Here, the modeler is concerned with identifying the arguments and determining their types.



Develop IDEF4 Protocol Diagrams

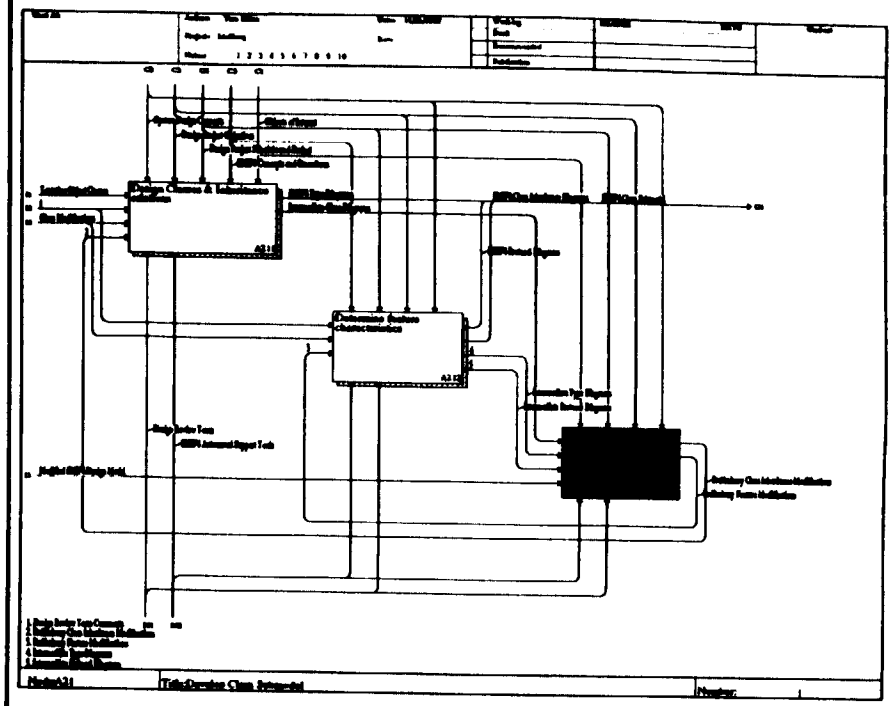
The Protocol Diagram maintains the argument list information for a routine feature. Each protocol diagram has two default arguments: self and return. Self refers to an instance of the class for which the feature is defined. Return refers to the object that is returned by the feature. Every argument specified in the Protocol is linked with the appropriate type link to the class that represents that arguments value type.



ORIGINAL PAGE IS
OF POOR QUALITY

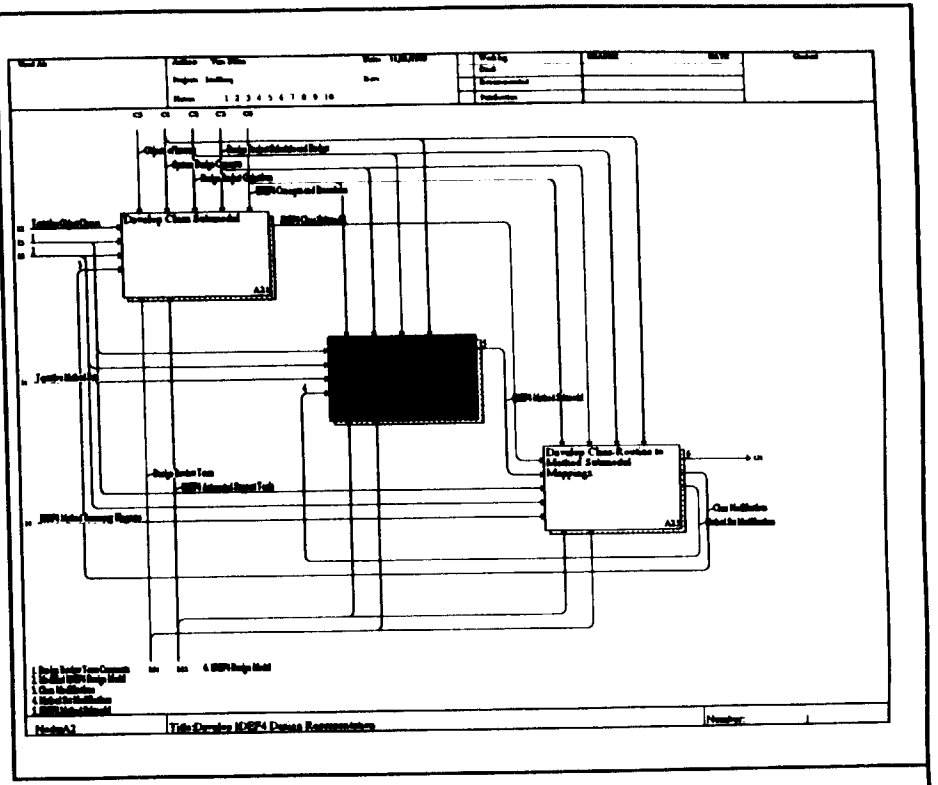
Eval Resulting Class/Feature Elements & Structures

Developing a class submodel is an iterative process. This activity is an analsys phase of the design development process. The existing classes are examined to determine if the objectives of the design are going to be met by an implementation built on these existing classes.



Develop Method Submodel

The Class Submodel represents the data associated with each class. The Method Submodel represents the functionality that must be satisfied by an implementation of the design. The structure to capture this functionality is the method set and dispatch mapping. The dispatch mapping maps a class-routine pair to a method set and a method set defines a set of contracts that must be satisfied by the class-routine pair.



ORIGINAL PAGE IS
OF POOR QUALITY

Used At:

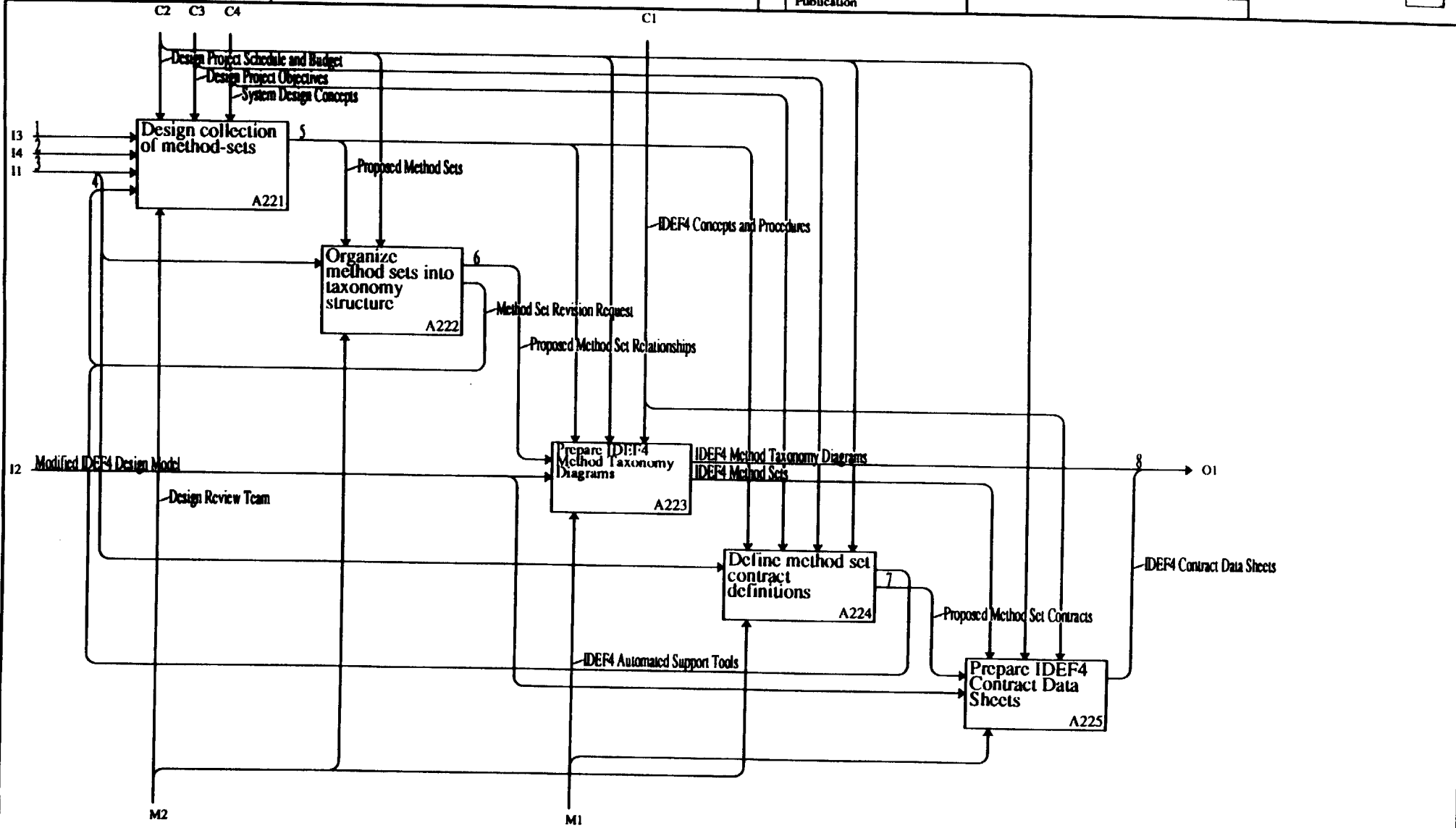
Author: Tom Blinn Date: 11/21/1989

Project: Idef4req Rev:

Notes: 1 2 3 4 5 6 7 8 9 10

	Working	READER	DATE
	Draft		
	Recommended		
	Publication		

Context

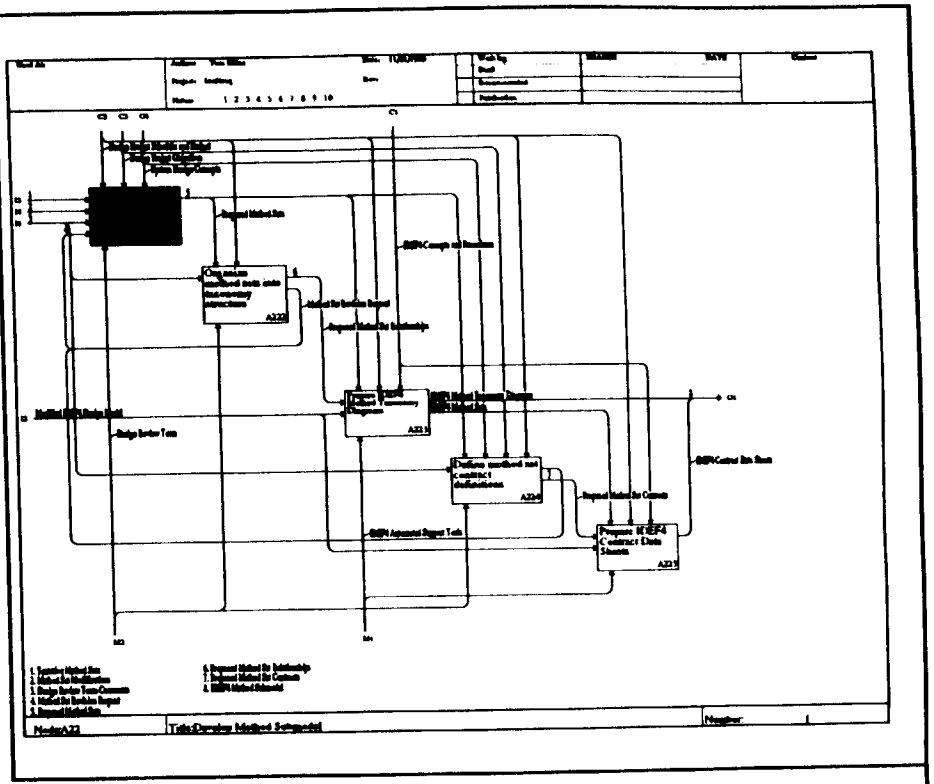


1. Tentative Method Sets
2. Method Set Modifications
3. Design Review Team Comments
4. Method Set Revision Request
5. Proposed Method Sets
6. Proposed Method Set Relationships
7. Proposed Method Set Contracts
8. IDEF4 Method Submodel

Node:A22 Title:Develop Method Submodel Number: 43

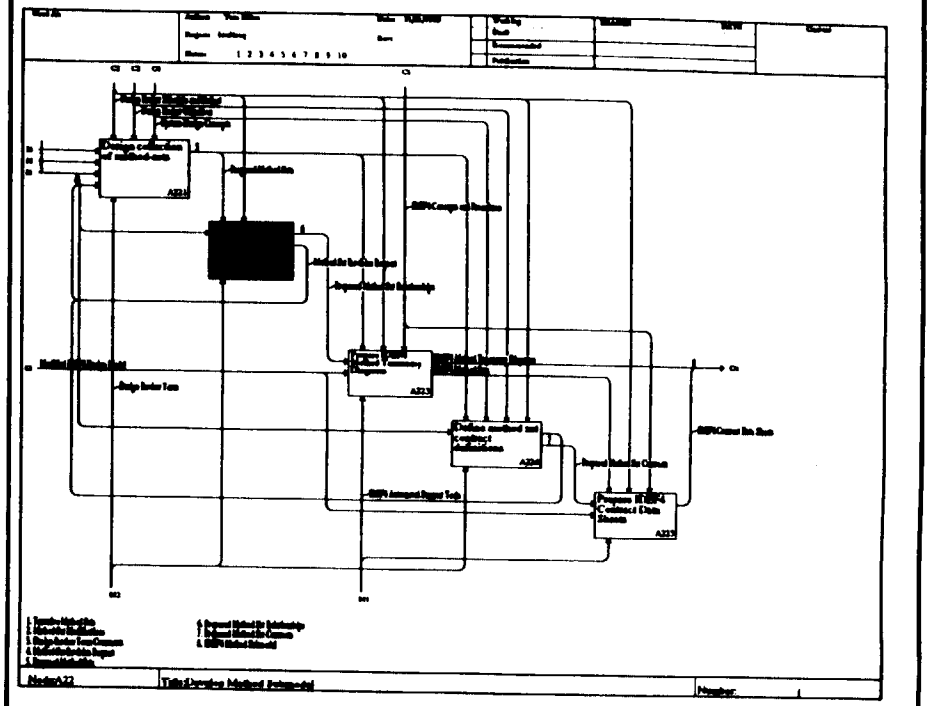
Design collection of method-sets

Here, the designer defines the method sets that will provide the necessary functionality to meet the design goals of the system.



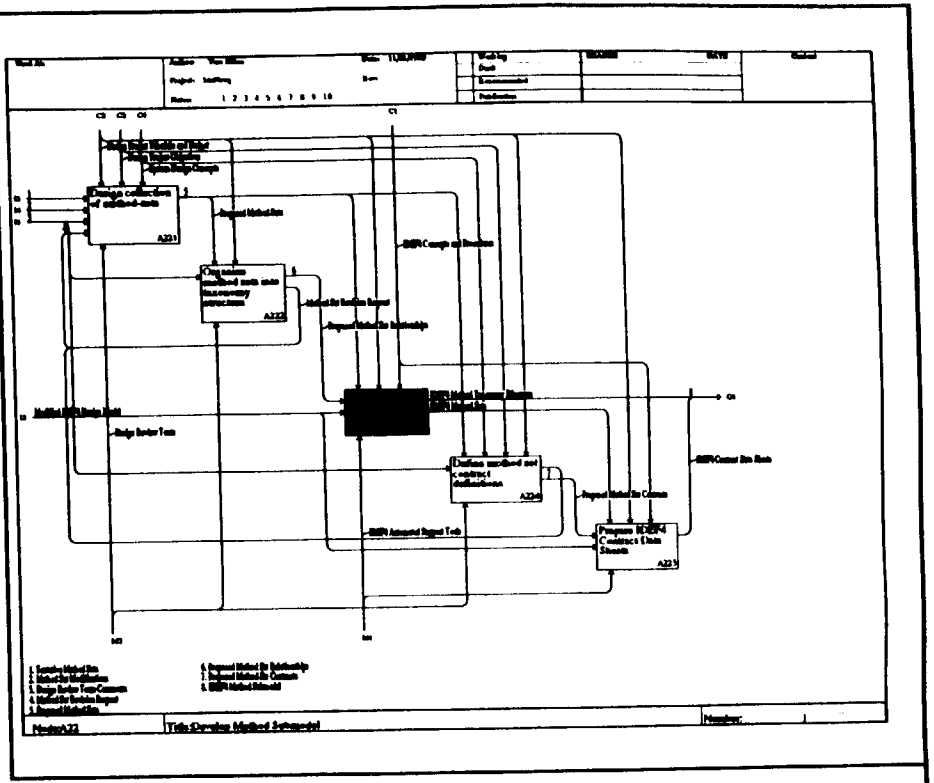
Organize method sets into taxonomy structure

Once the method sets have been defined, additional information can be captured by defining relationships between method sets. These relationships result in a taxonomy of related sets. For extensively studied method sets, these taxonomies can be quite useful.



Prepare IDEF4 Method Taxonomy Diagrams

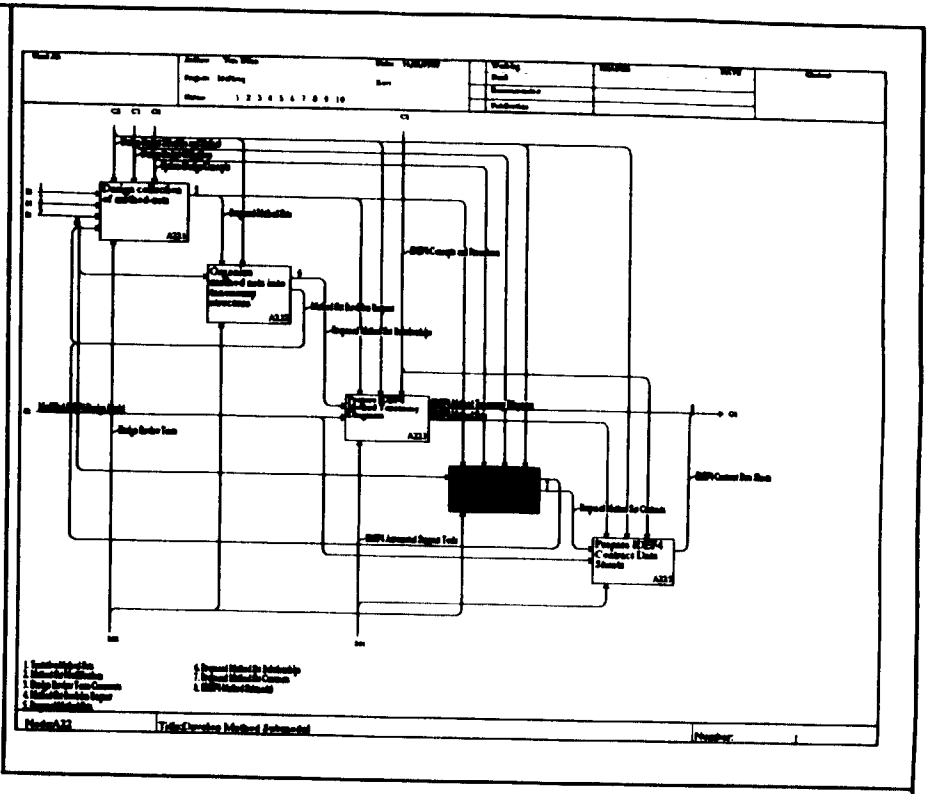
The Method Taxonomy Diagram is IDEF4's method at capturing the taxonomy of the method sets. The relationships represented in the Method Taxonomy Diagrams are pure superset/subset relationships, pointing from the superset to subset.



ORIGINAL PAGE IS
OF POOR QUALITY

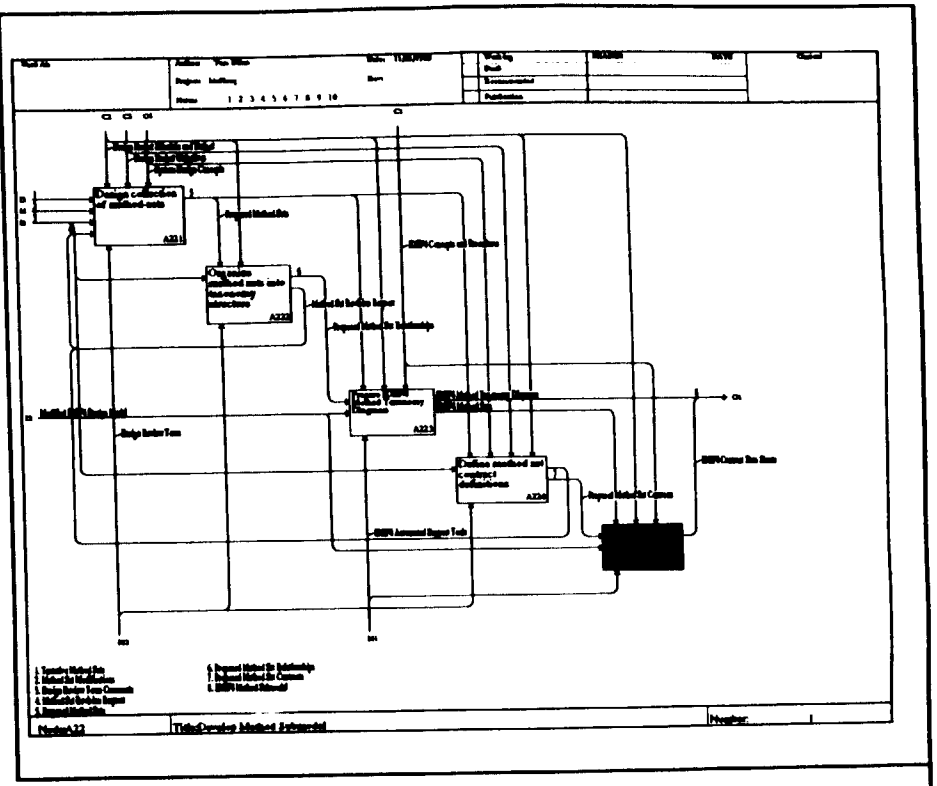
Define method set contract definitions

Once a method set has been defined, the requirements of the method set must be specified. A contract is a requirement that some method within the method set must satisfy. Here, the various contracts for the methods in the method sets are defined.



Prepare IDEF4 Contract Data Sheets

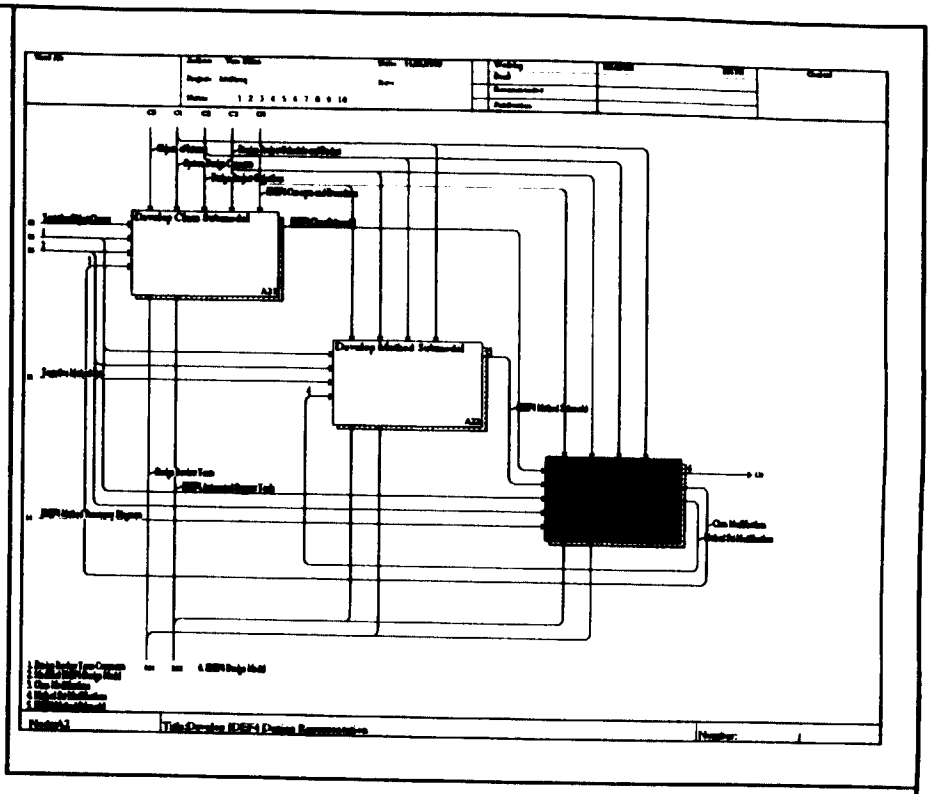
The contract requirements are maintained on a Contract Data Sheet. Each method set has its own Data Sheet. The contracts are textual descriptions of the constraints that must be satisfied by the method set.



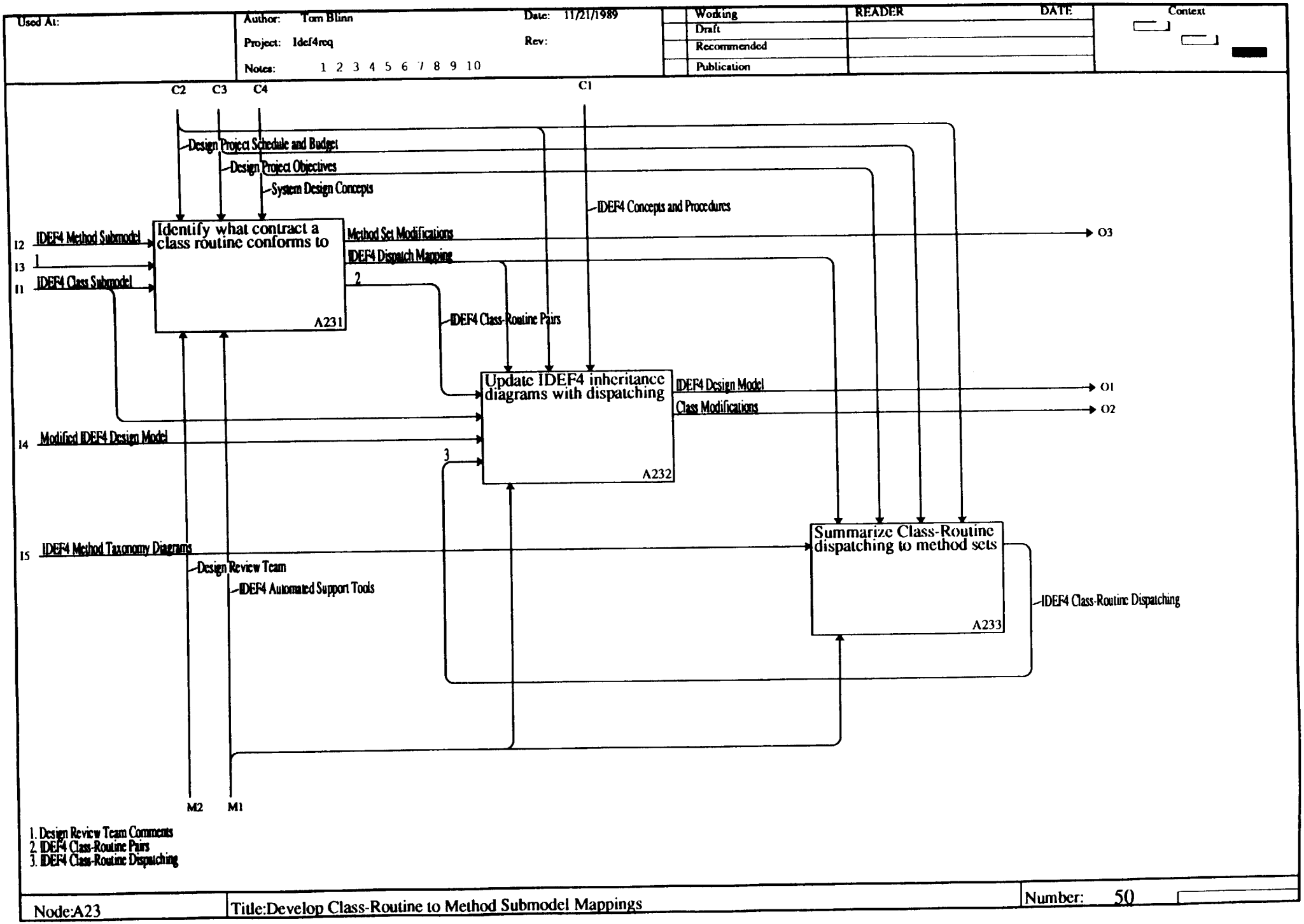
ORIGINAL PAGE IS
OF POOR QUALITY

Develop Class-Routine to Method Submodel Mappings

After the Class and Method Submodels have been defined, the two must be linked together. This is accomplished with a series of mappings between class-routine pairs and method sets.

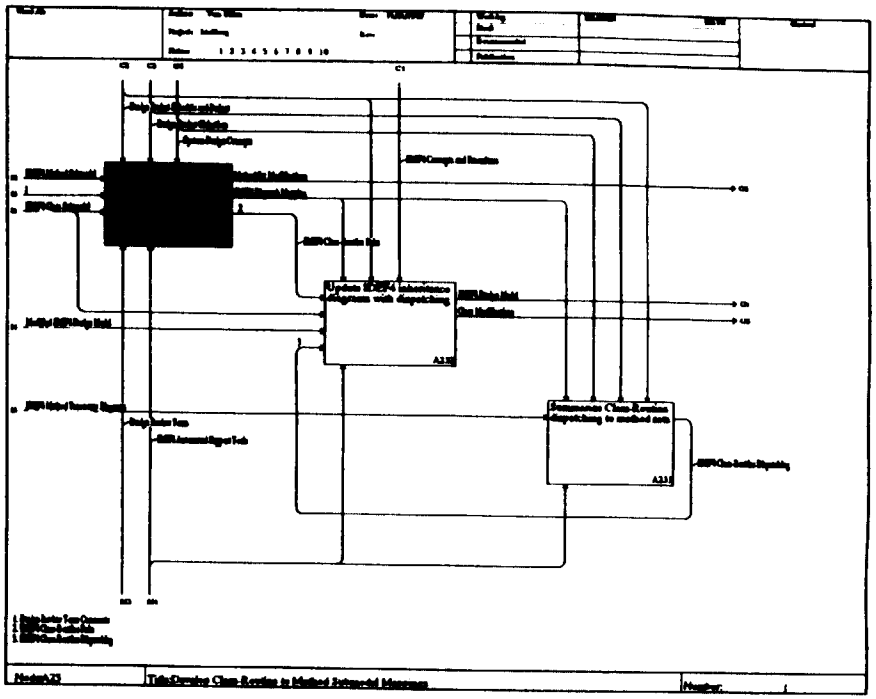


ORIGINAL PAGE IS
OF POOR QUALITY


Used At:
Author: Tom Blinn
Date: 11/21/1989
Project: Idef4req
Rev:
Notes: 1 2 3 4 5 6 7 8 9 10
Working
Draft
Recommended
Publication
READER
DATE
Context
C2
C3
C4
C1
Design Project Schedule and Budget
Design Project Objectives
System Design Concepts
IDEF4 Concepts and Procedures
12 IDEF4 Method Submodel
13 1
11 IDEF4 Class Submodel
Identify what contract a class routine conforms to
A231
Method Set Modifications
IDEF4 Dispatch Mapping
2
IDEF4 Class-Routine Pairs
O3
Update IDEF4 inheritance diagrams with dispatching
A232
IDEF4 Design Model
Class Modifications
O1
O2
14 Modified IDEF4 Design Model
3
Summarize Class-Routine dispatching to method sets
A233
15 IDEF4 Method Taxonomy Diagrams
Design Review Team
IDEF4 Automated Support Tools
IDEF4 Class-Routine Dispatching
M2
M1
1. Design Review Team Comments
2. IDEF4 Class-Routine Pairs
3. IDEF4 Class-Routine Dispatching
Node:A23
Title:Develop Class-Routine to Method Submodel Mappings
Number: 50

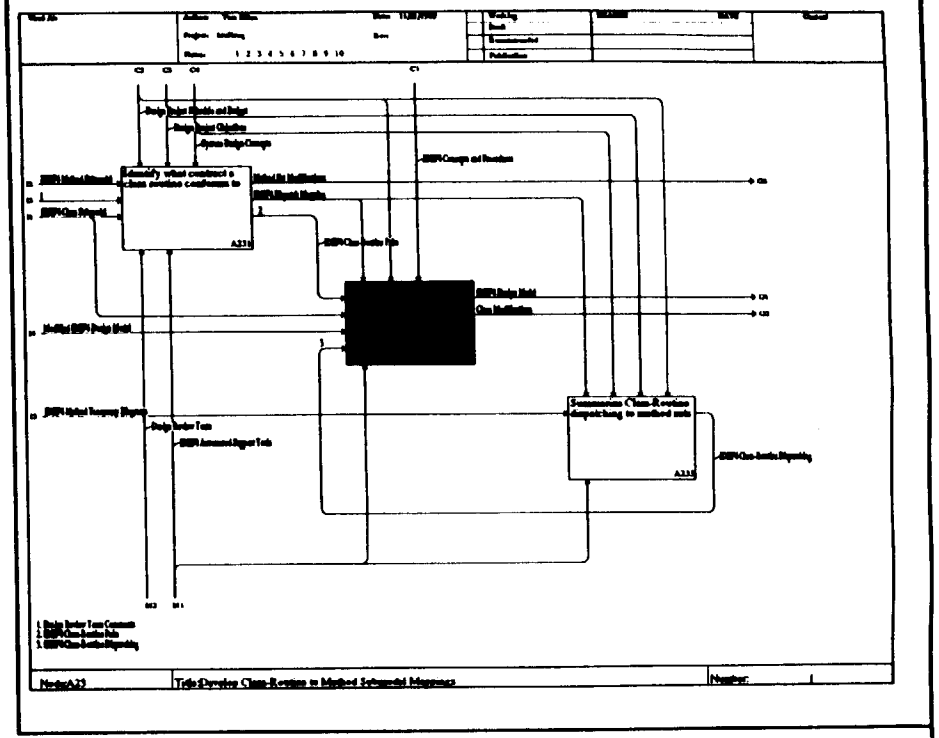
Identify what contract a class routine conforms to

Before a mapping can defined, the designer must determine what method set the routine should be associated with. This would involve browsing the existing method set defininitions to find the contracts that most closely match the intended functionality of the routine feature.



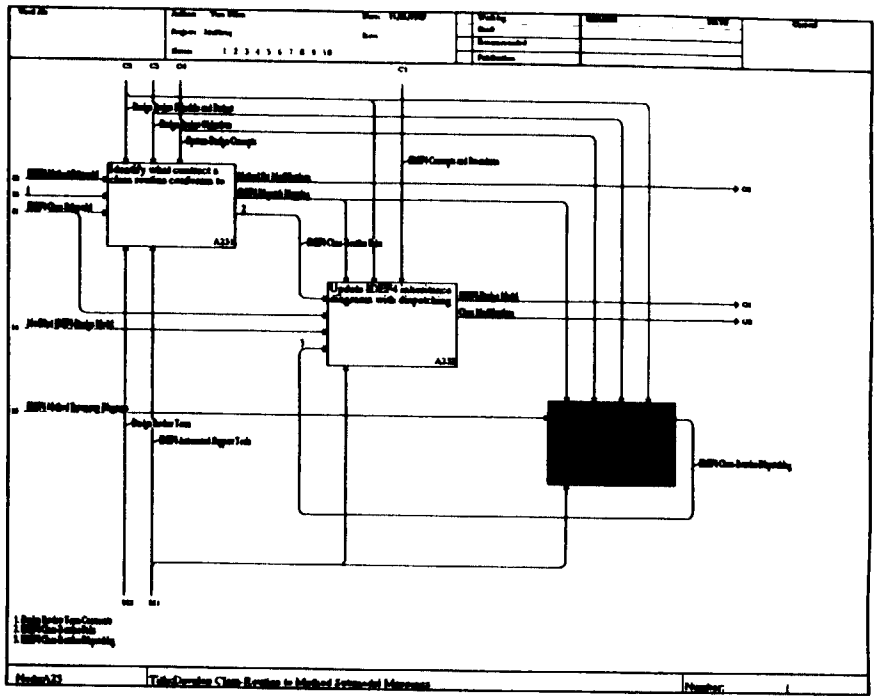
Update IDEF4 inheritance diagrams with dispatching

After the class-routine method mapping has been defined, it is necessary to update the inheritance diagrams to reflect the new mappings. This would involve including in brackets after the routine of a class the name of the method set to which the routine maps.



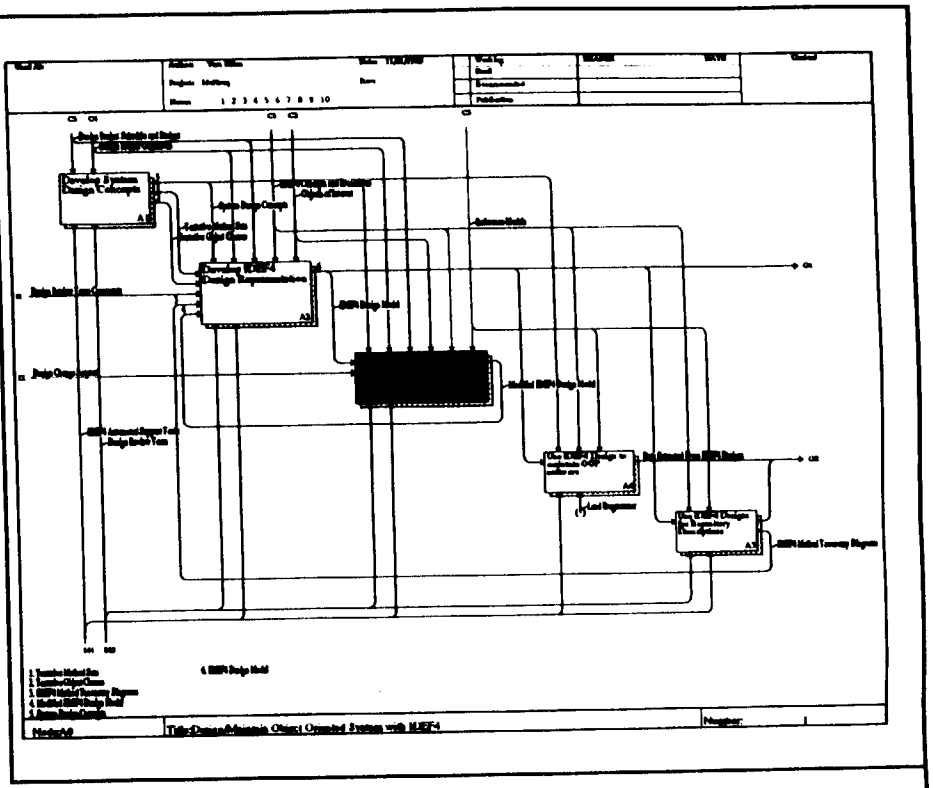
Summarize Class-Routine dispatching to method sets

Once the appropriate method sets have been identified for class-routine pairs, a mapping between the method set and class-routine pair must be specified. This is accomplished by attaching the class-routine pair to the method set.



Evolve IDEF4 Design

Once a design has been completed, it may become necessary to make minor modifications to the design model as an implementation of the design proceeds. The modifications must be minor, otherwise the effect on the implementation would be too great.



Used At:

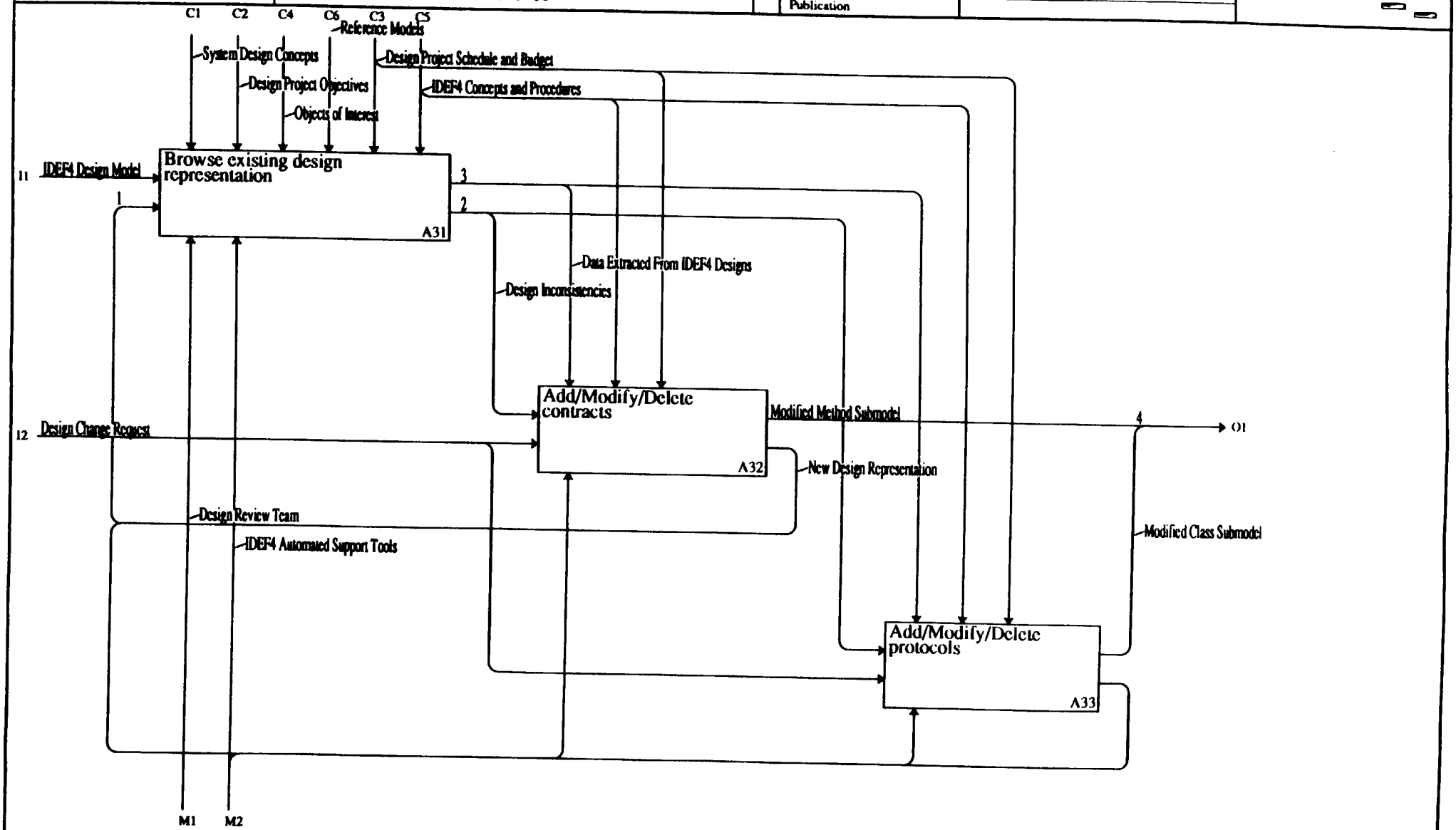
Author: Tom Blinn Date: 11/21/1989

Project: Idef4req Rev:

Notes: 1 2 3 4 5 6 7 8 9 10

	Working	READER	DATE
	Draft		
	Recommended		
	Publication		

Context

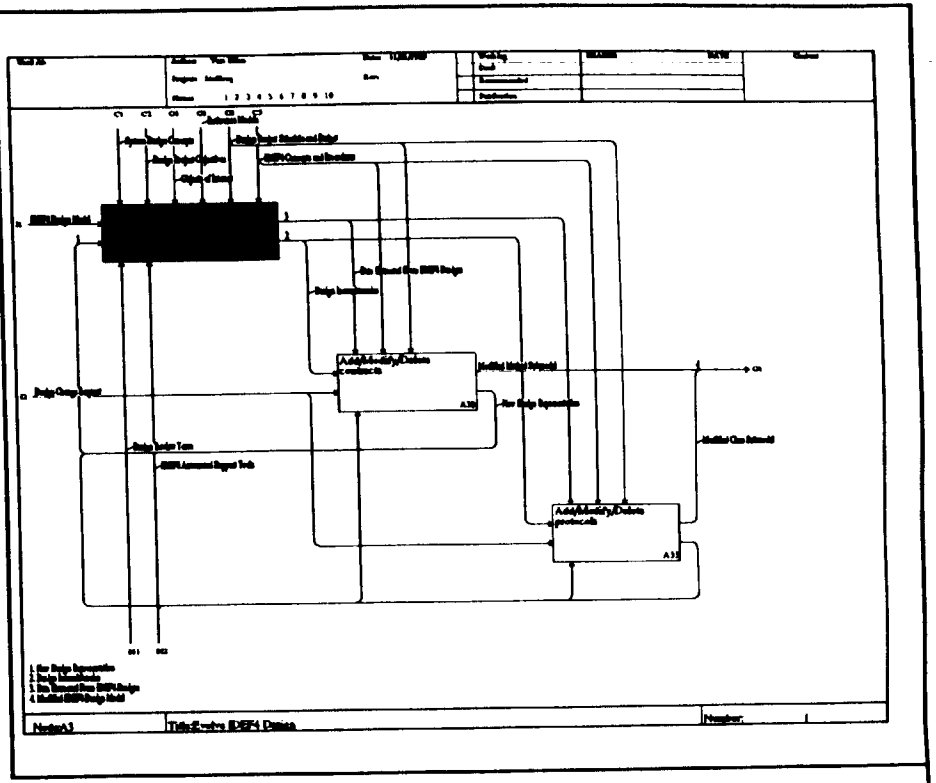


1. New Design Representation
2. Design Inconsistencies
3. Data Extracted From IDEF4 Designs
4. Modified IDEF4 Design Model

Node:A3 Title:Evolve IDEF4 Design Number: 55

Browse existing design representation

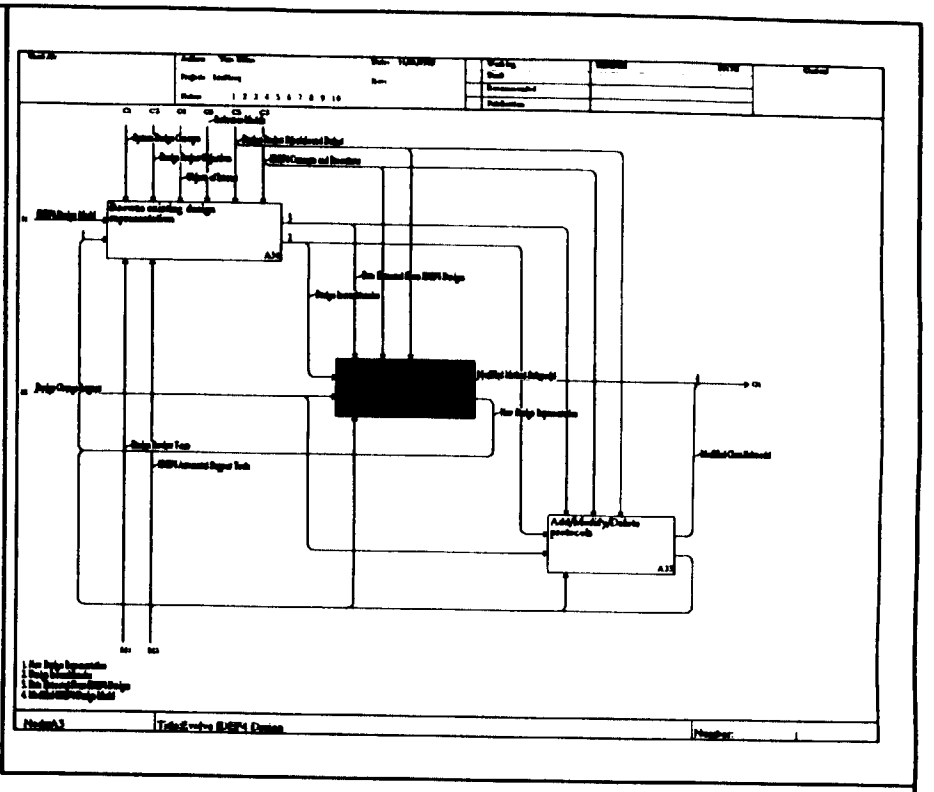
Here, the designer is examining the existing design for areas that might need modification or extension to resolve problems in the design that were discovered during the implementation of the design.



ORIGINAL PAGE IS
OF POOR QUALITY

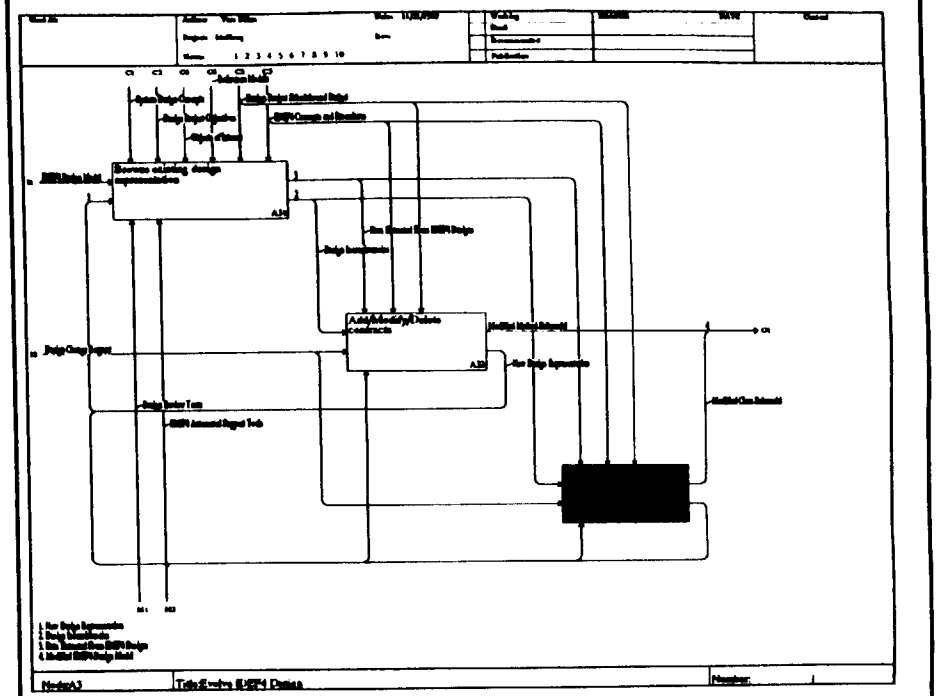
Add/Modify/Delete contracts

It may be necessary to revise the contracts associated with various method sets in the design model. This would result in a slight change in the functionality of the implementation.



Add/Modify/Delete protocols

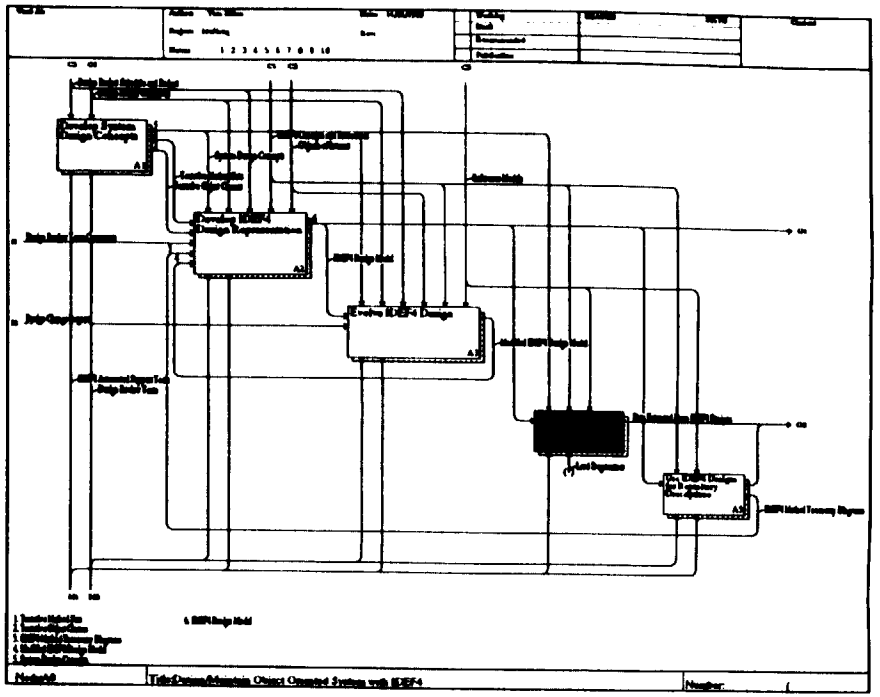
It may be necessary to revise the protocols associated with routine features. Certain arguments may no longer be necessary or more arguments may be required than originally expected.



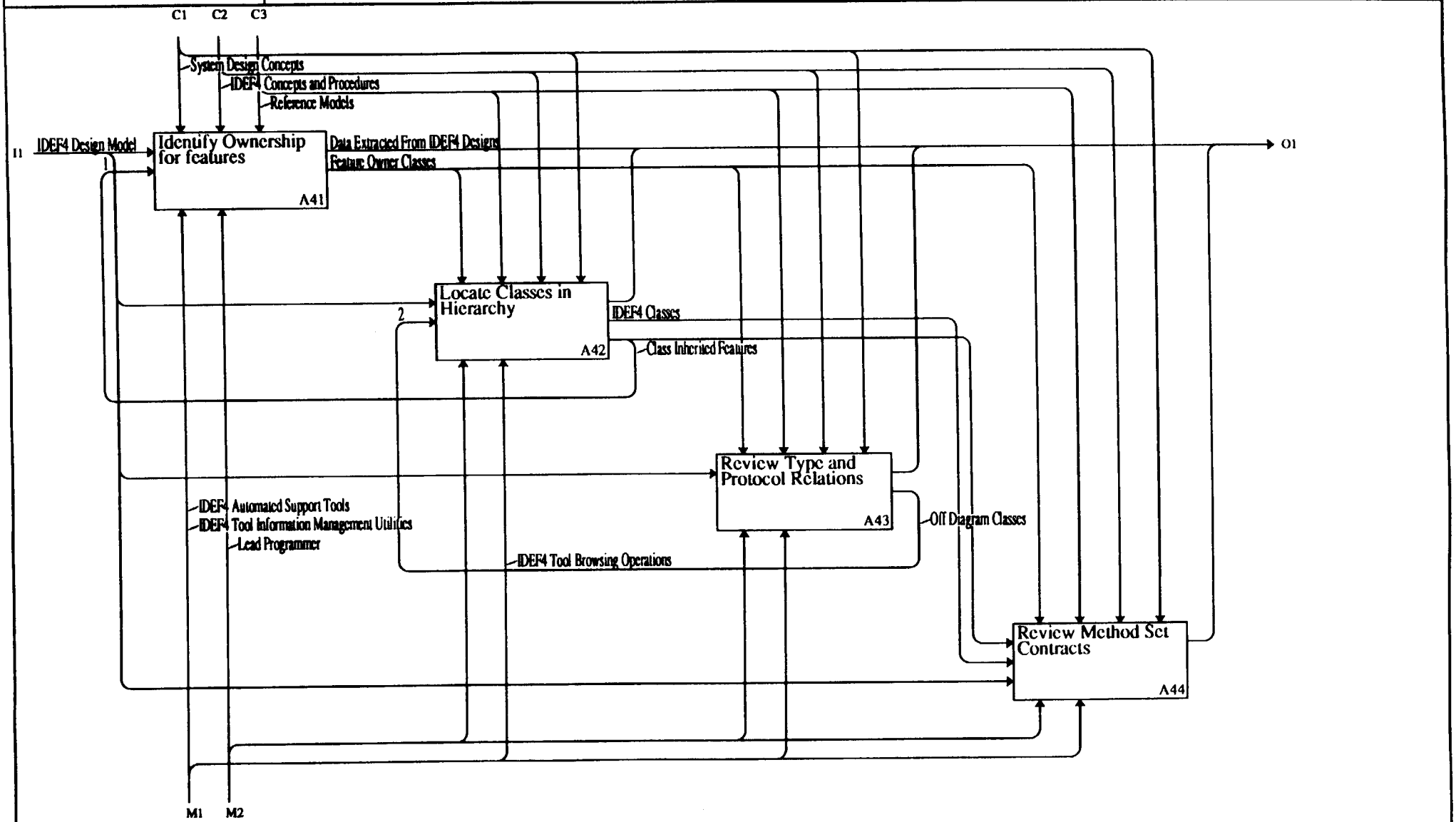
ORIGINAL PAGE IS
OF POOR QUALITY

Use IDEF4 Design to maintain OOP software

Once a design has been implemented, the design should still be retained so the the program can be updated in the future. If the lead programmer can access the design model, they can best determine where the changes should be made in the system to correct bugs or provide new functionality.



Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef4req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

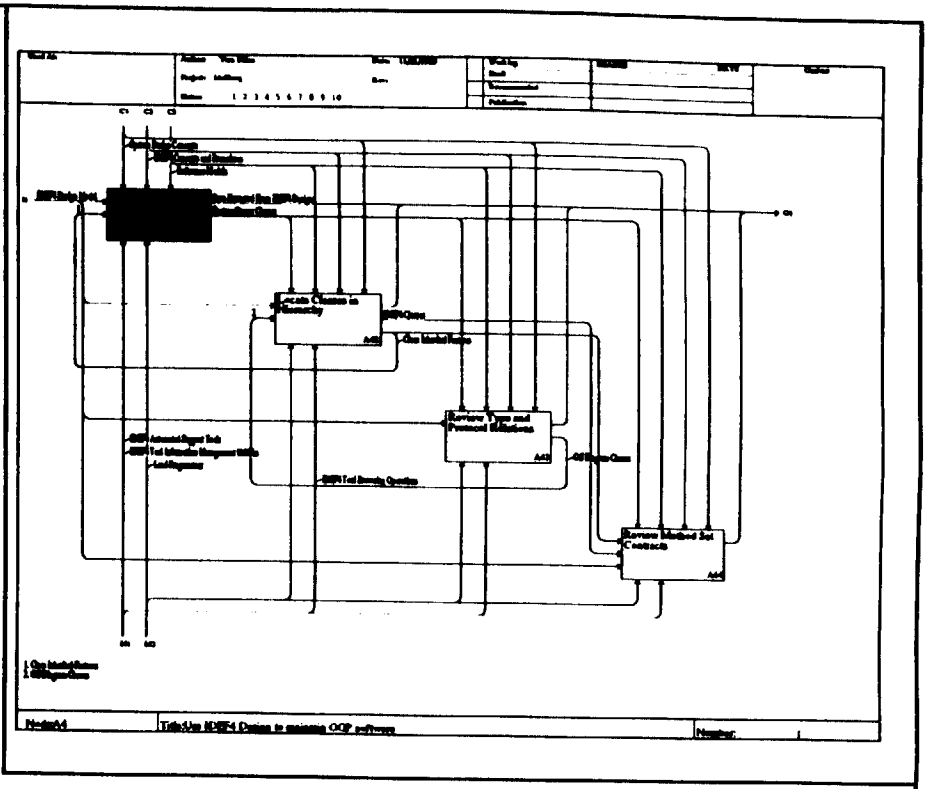


1. Class Inherited Features
2. Off Diagram Classes

Node:A4	Title:Use IDEF4 Design to maintain OOP software	Number: 60

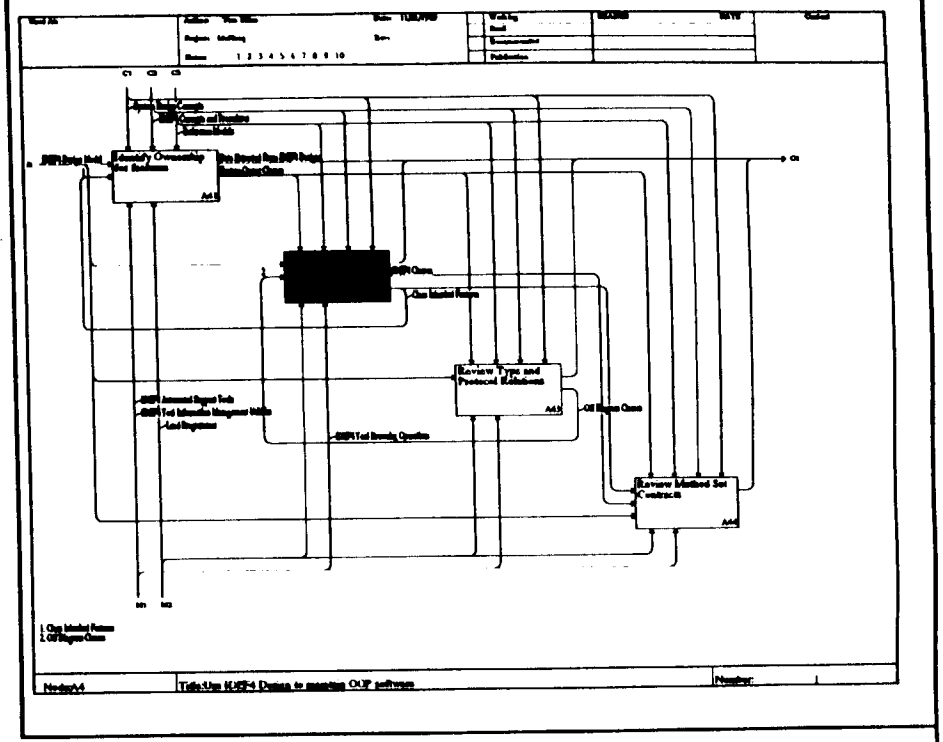
Identify Ownership for features

At this point, the programmer is attempting to determine where features are defined (i.e., which classes define that feature). The tool should take advantage of its Information Management Utilities to provide the various cross references necessary to locate rapidly the owner classes.



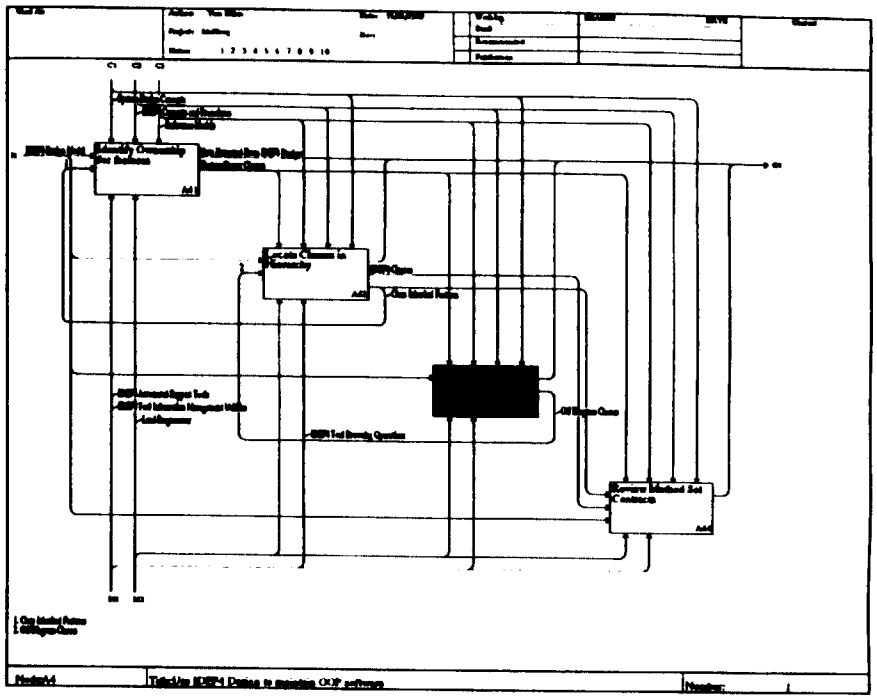
Locate Classes in Hierarchy

To gain a better understanding of a class, it would be useful to examine the inheritance relationships of the class. The programmer would locate the class in the various diagrams to gain an understanding of where the features of the class are defined. Browsing utilities should be provided to assist in this class location.



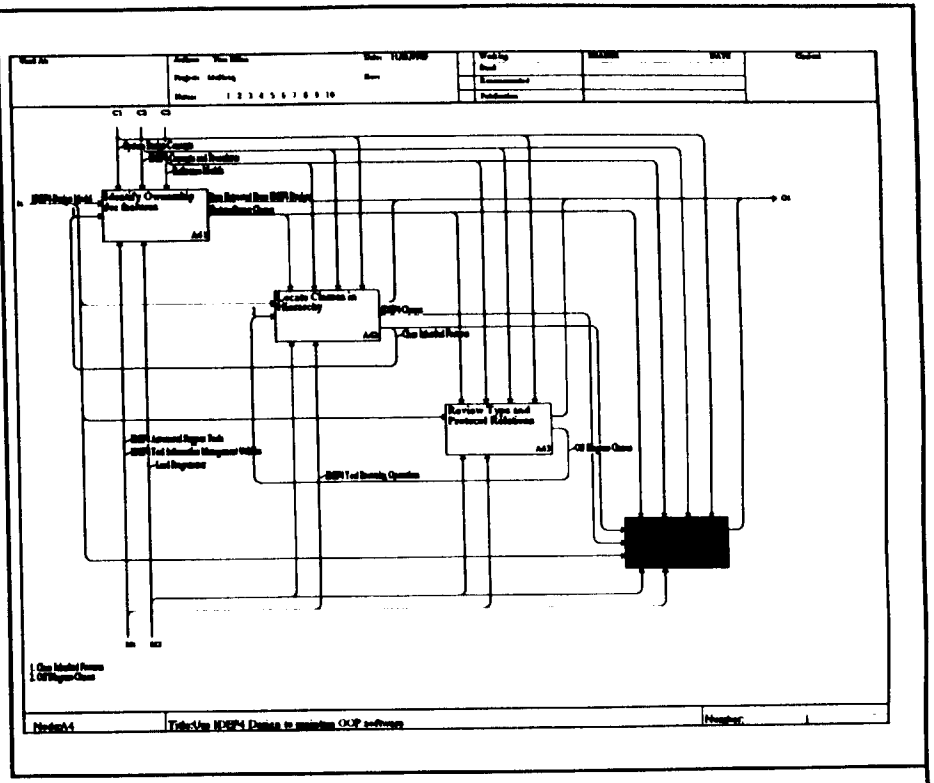
Review Type and Protocol Relations

At this point, the programmer examines the types of a feature as well as the types of the arguments to that features protocol. This would provide the ability to determine if the types in the design agree with the types that are used in the implementation.

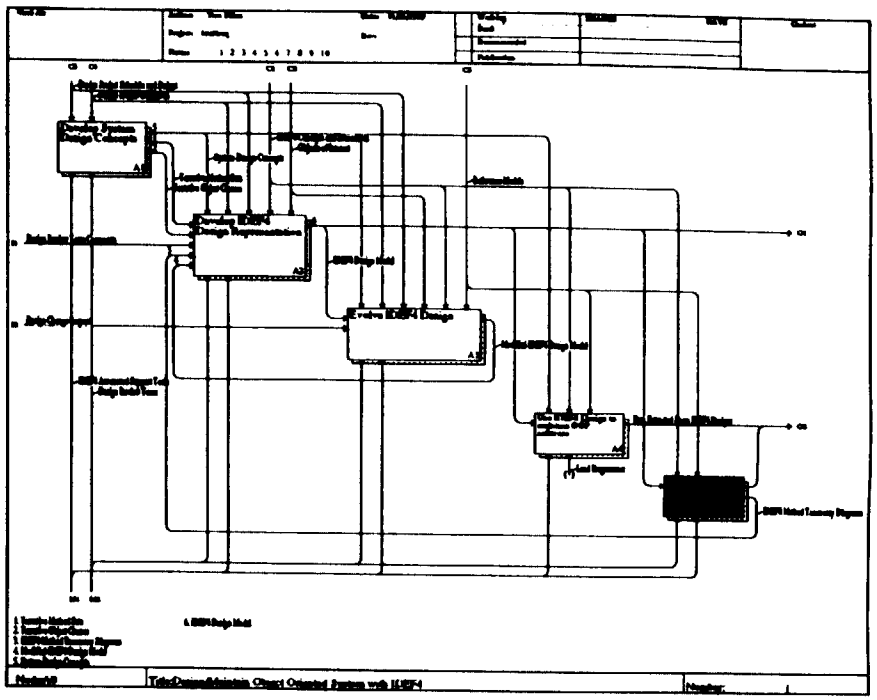


Review Method Set Contracts

This activity would involve examining the various contracts in the contract data sheet to ensure that the implementation of the system does in fact meet the requirements of the contracts. The Feature Owner Classes, along with the feature, provide the mapping to the method set.

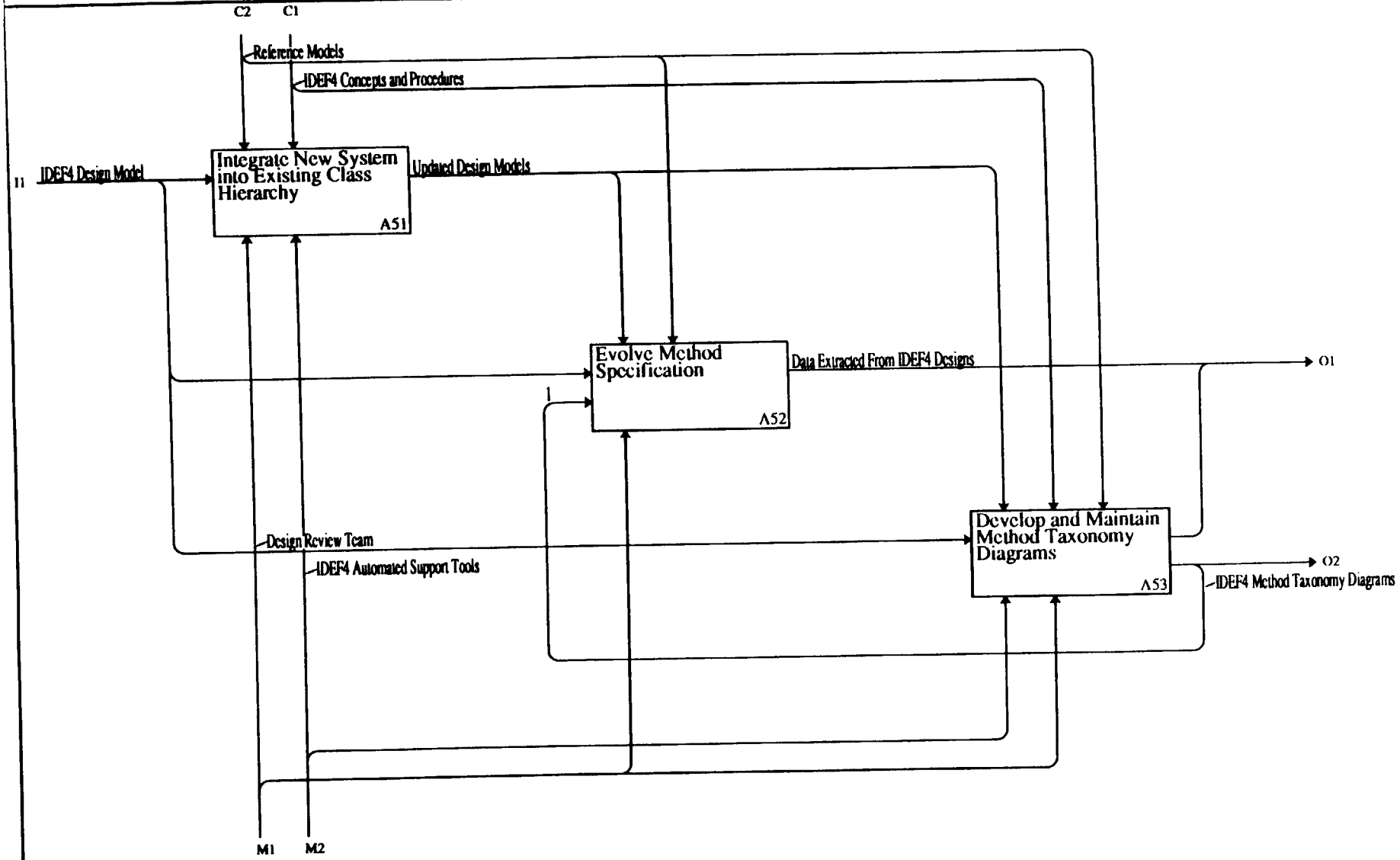


Use IDEF4 Designs for Repository Descriptions



--This space intentionally left blank--

Used At:	Author: Tom Blinn	Date: 11/21/1989	Working	READER	DATE	Context
	Project: Idef4req	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

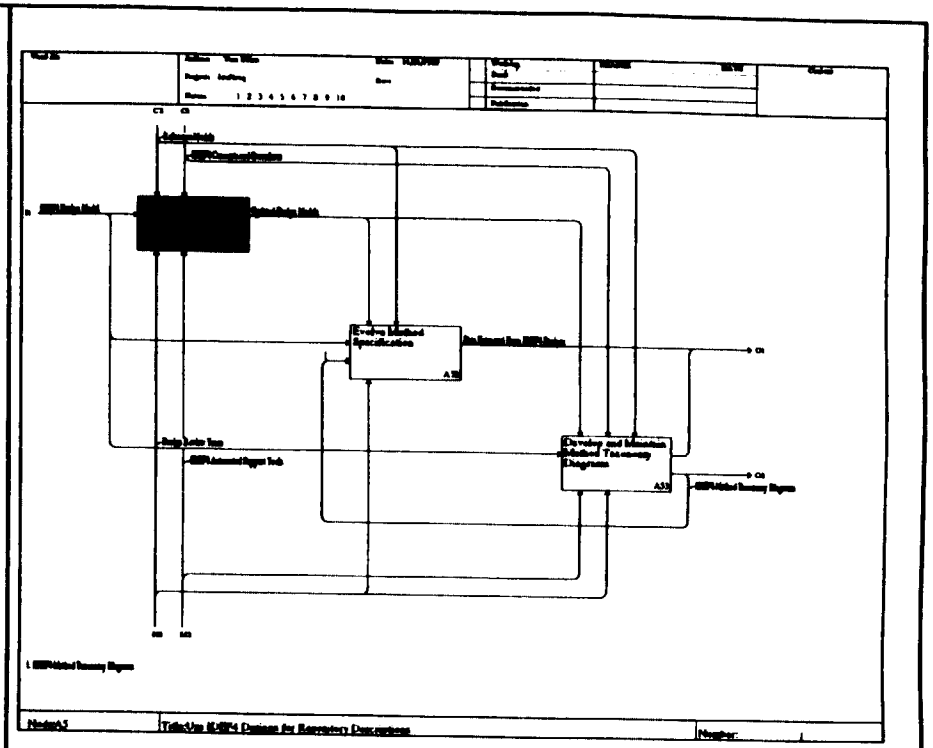


1. IDEF4 Method Taxonomy Diagrams

Node:A5	Title:Use IDEF4 Designs for Repository Descriptions	Number: 66

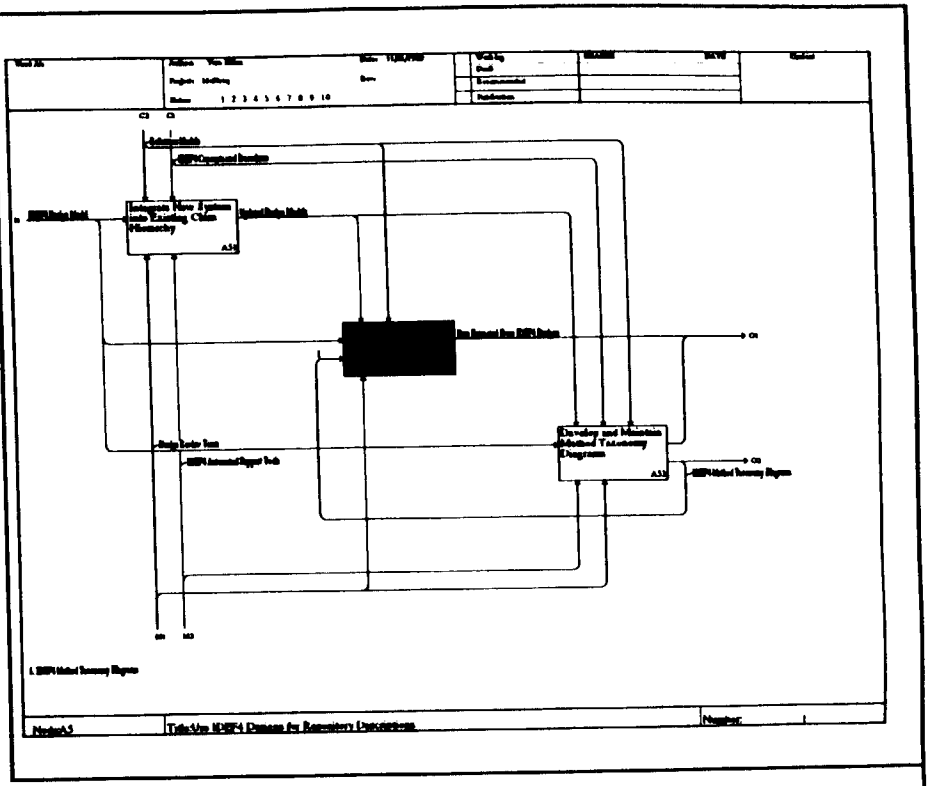
Integrate New System into Existing Class Hierarchy

This activity is intended to promote the modularity of object-oriented systems. By developing classes as modules, is should require a minimal effort to include a new system into an existing class hierarchy. This would allow the designs of two existing systems to be incorporated into one larger system with overlapping functionality removed.



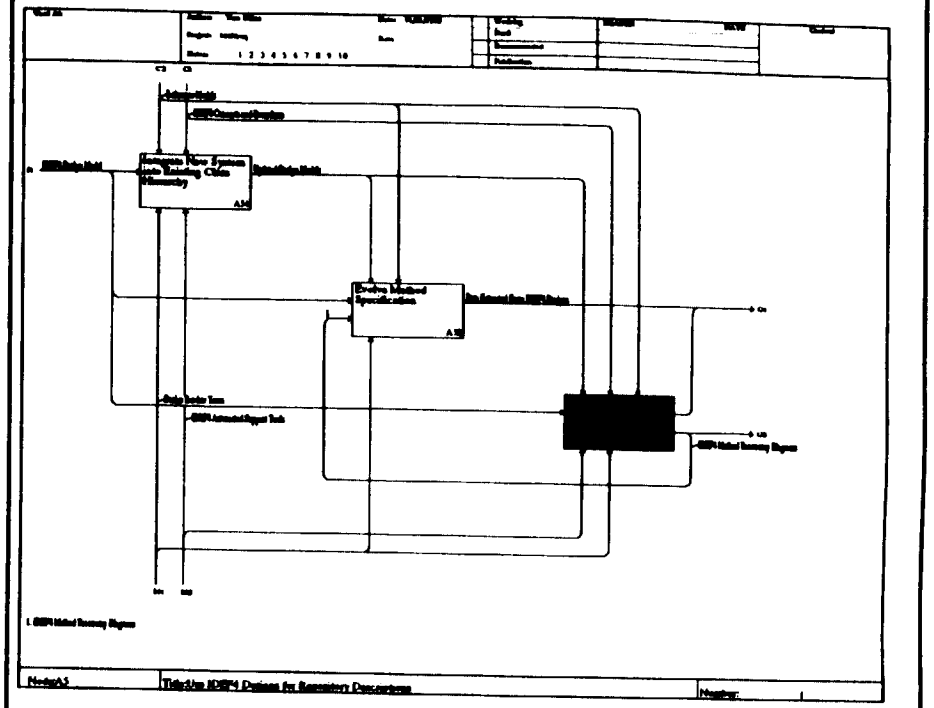
Evolve Method Specification

This is almost an extension to IDEF4 in that the method set can be manipulated to produce individual methods that satisfy the contracts of the method set. This approaches more of an implementation aspect than a design aspect, except that no code will be produced. Just a further breadown of the method set into smaller functional units.



Develop and Maintain Method Taxonomy Diagrams

As more is learned about the various method sets and their relationships to each other during the implementation of the design, it is important that the taxonomy diagrams be updated to capture this valuable knowledge.



GLOSSARY

Class Descriptions

A class description is a textual description of the information that is maintained by instances of that class.

Class Inherited Features

Inherited Features are those features that have been attached to a class through an inheritance link with another class.

Class Modifications

Since the Class and Method Submodel are developed somewhat independently, it is only natural that when mappings are generated between the two models some modifications to the class structures. This concept represents the changes that are required as a result of the dispatch mapping.

Data Extracted From IDEF4 Designs

Sometimes information from IDEF4 designs can serve as valuable input to new design models. In these cases, it is necessary to browse and extract information from an existing design for use in a new design.

Data extracted from Requirements Analysis

The development of any system begins with a Requirements document. As a result, the initial design step is to examine the requirements and extract the information from the document that will effect the design of the system.

Design Change Request

No design is ever correct the first time. As a result, it is often necessary to make modifications to the design. A design change request provides a formal channel for making modificatons to the design so that the impact of such a change will be minimized.

Design Inconsistencies

As designs grow large, it is also possible that those designs might become inconsistent.

Design Project Objectives

This concept represents the general goals of the design project.

Design Project Schedule and Budget

This concept represents the time and monetary constraints that exist for the design process.

Design Review Team

The Design Review Team are the people responsible for the design. It is expected that a portion of the design is completed and then submitted to the team for examination.

Design Review Team Comments

These comments are the main thrust of the design process. Every part of the design must undergo scrutiny by the review team. The team's comments will have a critical effect on the design of the system.

Feature Arguments

Certain features that require computation to determine that feature's value can have arguments that provide data for the generation of the value. These arguments are displayed in the Protocol Diagram for the feature.

Feature Owner Classes

A feature can be owned by several classes. This concept represents the classes that all have a particular feature in common.

Feature Return Type

A feature can either return or take a value when queried. The type of the value is specified by the Feature Return Type.

Feature Type Relations

These relations refer to one of the four different type relationships that can be established for a feature. See the Type Links in IDEF4 Requirements Section.

Functional Requirements

The functional requirements outline all the functions that a software system must

provide.

IDEF4 Automated Support Tools

The tools provide the functionality required to produce object-oriented designs for software systems based on the IDEF4 design method.

IDEF4 Class Inheritance Diagrams

The Class Inheritance Diagrams present the superset/subset relationships between classes in the design model. The arrow between two classes in the diagram points from the parent to the child. The child inherits all features that the parent owns.

IDEF4 Class Invariant Data Sheet

Each class has an Invariant Data Sheet associated with it. This sheet captures in textual format the required behavior that all instances of the class must exhibit at all times.

IDEF4 Class Submodel

The Class Submodel maintains all the information pertaining to classes, features, and the protocols of the features as well as the feature type and class inheritance relationships.

IDEF4 Class-Routine Dispatching

This operation links method sets with a class-routine pair. In effect, this dispatching provides the bridge between the class and method submodels.

IDEF4 Class-Routine Pairs

A class routine-pair is simply a relationship between a class and a feature. The pairing indicates that the feature is owned by the class.

IDEF4 Classes

IDEF4 Classes are the main structures of the design model. The represent the structure of the objects that will exist within the software system.

IDEF4 Concepts and Procedures

The Concepts and Procedures refer to the syntax, semantics, and strategies of the IDEF4 design method.

IDEF4 Contract Data Sheets

The Contract Data Sheets maintain the contracts specified for a method set. A contract specifies some function that a method that is a member of the method set must perform.

IDEF4 Design Model

An IDEF4 Design Model contains the class definitions and method set definitions that make up the design for an object-oriented software system. The design model also maintains information on class features and their types.

IDEF4 Dispatch Mapping

The Dispatch Mapping provides the links between the Class Submodel and the Method Submodel by link class-routine pairs with method sets.

IDEF4 Feature

A feature is a characteristic of a class and can play one of five roles:
1) attribute - can be either a slot or a function.
2) slot - references a value.
3) function - returns a value.
4) routine - a routine forces the execution of code (either
a procedure or function.
5) procedure - a side-effecting routine.

IDEF4 Method Set Relationships

The relationships are the links between method sets in a method taxonomy diagram. The relationship specified is the pure subset/superset relationship.

IDEF4 Method Sets

Method Sets define the functionality of the system by maintaining the contracts that must be satisfied by methods in the method set.

IDEF4 Method Submodel

The Method Submodel maintains all information on method sets and their contracts, as well as client diagrams that present the caller/callee relationship between routines.

IDEF4 Method Taxonomy Diagrams

The Method Taxonomy Diagrams present the subset relationships between method sets maintained in the Method Submodel.

IDEF4 Protocol Diagrams

The Protocol Diagrams present the argument lists of routine features. The expected types of the arguments are also displayed in this diagram.

IDEF4 Tool Browsing Operations

The Tool Browsing Operations provide the ability to move around the various diagrams in the Design Model. This would involve examining classes, features and their types, as well as method sets.

IDEF4 Tool Information Management Utilities

The Information Management Utilities provide the ability to trace objects throughout the design model. For example, this utility would be able to determine all the classes that defined or inherited a particular feature.

IDEF4 Type Diagrams

The Type diagrams display the type relationships between features of classes. Four different types of links are defined and indicate the allowable value types that features can return or take.

Intermediate Class Diagrams

During the development of the class diagrams, the diagrams must be reviewed before they are accepted as part of the design. These pre-release diagrams are the Intermediate Class Diagrams.

Intermediate Protocol Diagrams

During the development of the protocol diagrams, those diagrams must be reviewed before they are accepted as part of the design. These pre-release diagrams are the Intermediate Protocol Diagrams.

Intermediate Type Diagrams

During the development of Type Diagrams, the diagrams must be reviewed before they are accepted as part of the design. These pre-release diagrams are the Intermediate Type Diagrams.

Invariant Class Requirements

These requirements are constraints on instances of classes that must be true for all instances at all times.

Lead Programmer

The Lead Programmer will have access to the design model so that questions concerning the implementation can be answered by browsing the design model.

Method Set Modifications

Method sets can be modified during the dispatch mapping. These modifications must be reflected in the method submodel.

Method Set Revision Request

This request provides a formal channel through which modifications to existing method sets can be requested.

Modified Class Submodel

As a design evolves, the class submodel will be modified several times.

Modified IDEF4 Design Model

As a design evolves, the design model undergoes changes. This involves changes to the Method Submodel as well as the Class Submodel.

Modified Method Submodel

As a design is evolving, the method submodel will be modified several times. This modified submodel must be maintained in the design.

New Design Representation

This is a new representation of the design as the design is evolving. It results from the addition of or modification to contracts or protocols.

Objects of Interest

The objects are entities that are to be manipulated by the software system that are of special interest to the design team. It may be that these objects may eventually become classes in the system design.

Off Diagram Classes

The off diagram classes are those classes that are referred to in a diagram but do not occur directly in the diagram. This reference would provide a link to the diagrams in which the class actually does appear.

Preliminary Class Inheritance Modifications

During the development of a model, faulty inheritance relationships may be determined. This situate will require preliminary modifications to correct the situation. These modifications are preliminary because they occur before the design has been completed.

Preliminary Feature Modifications

During the development of a design model, examination of the design may reveal unexpected feature characteristics in the class hierarchy. When these are encountered, preliminary modifications (preliminary because they occur before the design has been released) are required.

Proposed Method Set Contracts

Before the contracts for a method set can be completely defined, it may become evident that changes are necessary. Therefore, the original ideas on the contracts are just the proposed contracts, not necessarily the final contracts.

Proposed Method Set Relationships

Before all the method set relationships can be defined in the design, it may become obvious that the intended relationships will not provide an adequate design. Therefore, the actual relationships may change with respect to the intended relationships. For this reason, the original relationships are the proposed relationships, for they may change.

Proposed Method Sets

Before a Method Taxonomy can be developed, the design team must have an idea of what method sets are required. These method sets are not always implemented as they were originally intended so the intended method sets are labelled as proposed method sets.

Reference Models

These models are existing IDEF4 models that might provide valuable input to the design currently being developed.

Similar Existing Classes

These are classes that exist within IDEF4 design models that exibit similar characteristics. This may mean they have identical features or that the features of the class map to similar method sets that have almost identical contracts.

Similar Features

This represents the situation where the protocols and contracts are examined to determine the features that are similar to the features necessary in the new design. These features will provide reference to the classes that could be similar to one currently under design.

System Design Concepts

These concepts define the "look and feel" of the software system for which the design is being produced.

System Philosophy

The system philosophy will determine how the system will be partitioned as well as indicated what type of features should be supported. These features will determine which classes should be examined in the existing repository classes.

Tentative Method Sets

Tentative Method Sets precede the Proposed Method Sets. The tentative sets were indicated as method sets that might possibly be necessary in the method submodel while the proposed method sets are method sets that will exist in the method submodel at some point, but that may be removed later as the design evolves.

Tentative Object Classes

Tentative Object Classes precede the proposed object classes. The tentative classes are those classes that might be necessary upon an initial study of the system requirements. The proposed classes will appear in the design model at some point but may be deleted at a later date as the design evolves.

Updated Design Models

Updated design models are models that result from the modification or extension of existing design models.

[redacted] INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

Appendix D: IDEF1 Model of IDEF4

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE:	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

I.D. No.	Source Material Name	Received From
1	Idef4 Technical Paper	KBS Laboratory
2	IDEF3 Technical Report	KBS Laboratory

Node:	Title: Source Material Log	Number:

Used At:	Author: tom blinn	Date: 11-19-89			
	Project: IDEF4	Rev:			
	Notes: 1 2 3 4 5 6 7 8 9 10				

	Working	READER	DATE	Context
	Draft			
	Recommended			
	Publication			

I.D. No.	Source Data Name	Cross Reference Source Material

Node:	Title: Source Data Log	Number:

Used At:	Author: tom blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF4	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			

Entity Class I.D. No.	Entity Class Name	Source Data I.D. No.
1	Object Class	
2	Object Class Invariant	
3	Object-Class-Occ-as-Super-Class	
4	Method-Set-Specification	
5	Method-Set-Contract	
6	Feature	
7	Feature-Occurence-in-Object-Class	
8	Object-Class-Feature to Method-Set Mapping	
9	Attribute	
10	Routine	
11	Slot	
12	Function	
13	Procedure	

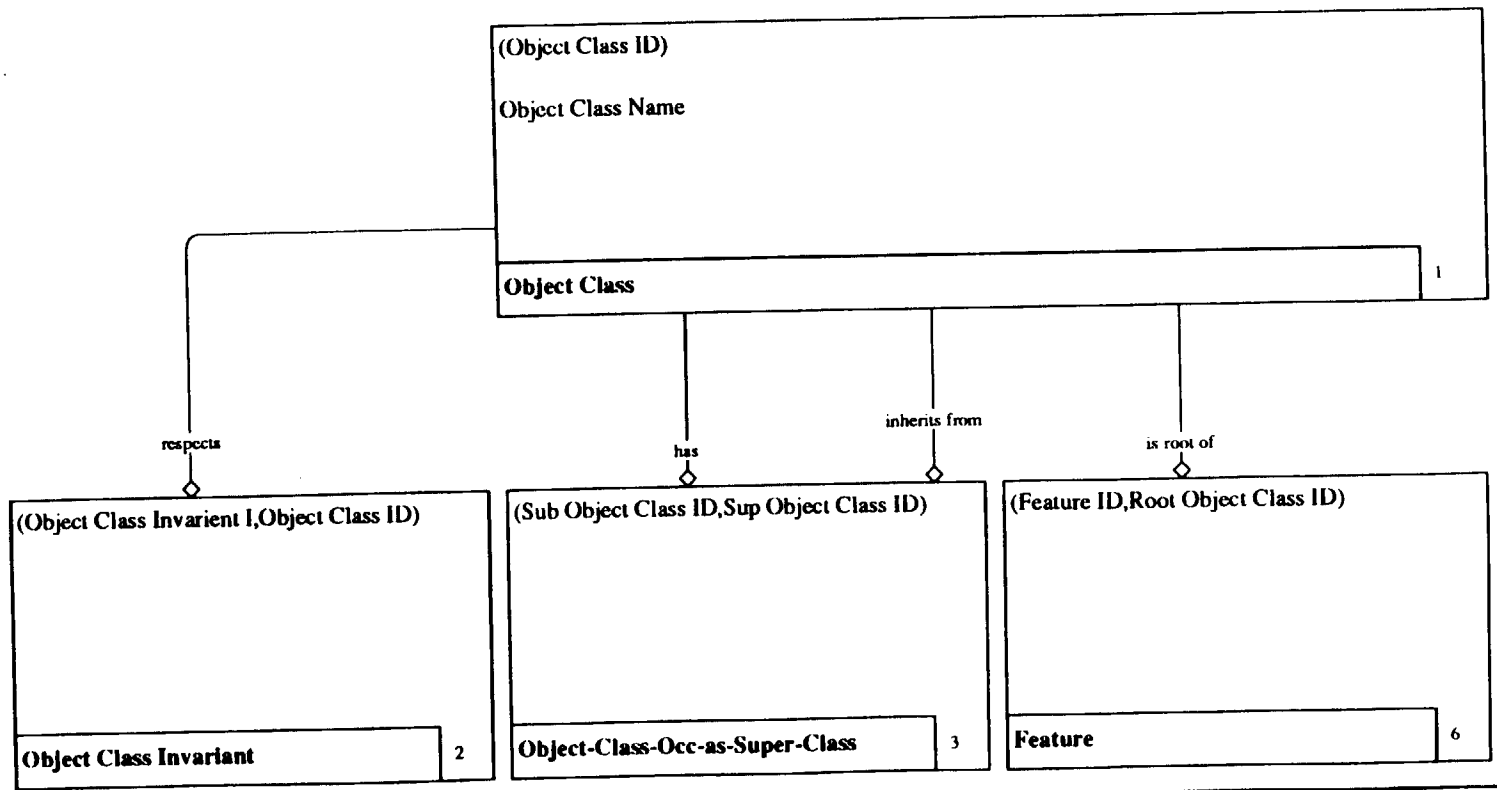
Node:	Title: Entity Class Pool	Number:

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE:	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			

I.D. No.	Attribute Class Name	Source Data I.D. No.
1	Object Class Name	
2	Object Class ID	
3	Feature Name	
4	Source Class	
5	Root class	
6	Feature type	
7	Feature redefinition type	
8	Feature occurence status	
9	Attribute Return type	
10	Method Set ID	
11	Method Set label	
12	Feature ID	
13	Object Class Invarient ID	
14	Object Class Invarient Specification	
15	Object Class Invarient Glossary	
16	Method Set Glossary	
17	Contract ID	
18	Contract Specification	
19	Contract Glossary	

Node:	Title: Attribute Class Pool	Number:

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE:	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



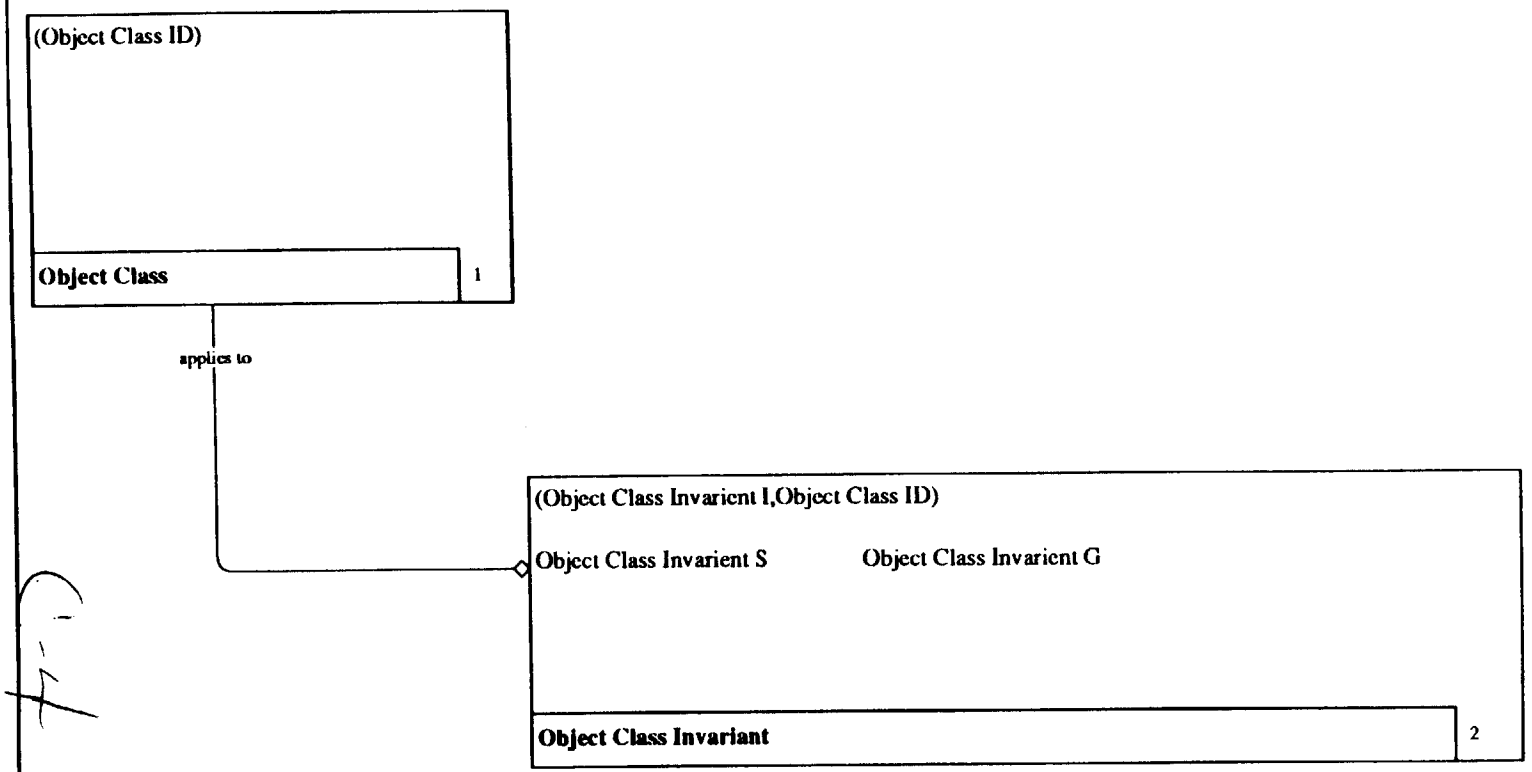
Node: E1	Title: Object Class	Number: 25

Entity Class Definition:
Key Classes:
(Object Class ID)
Owned Attribute Classes:

Name: Object Class ID
Definition: A unique non-reoccuring number assigned to an object class.

Name: Object Class Name
Definition: The unique name of an object class.

Used At:	Author: tom blinn	Date: 11-19-89		Working	READER	DATE	Context
	Project: IDEF4	Rev:		Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10			Recommended			
				Publication			



Node: E2	Title: Object Class Invariant	Number: 23

Entity Class Definition: The invariants of an object class are those propositions which are always true of every instance of the object class. An object class can own its' own set of invarients or it may just inherit the invarients of its super classes. Inheritance of invarients is identical to feature inheritance in IDEF4. Additional propositions in a subclass are combined with those of the superclasses using the principle of logical conjunction. In this model we could have chosen to represent the invariant as a text attribute of the object class. However, looking towards the future it is likely that there will be a formal language defined such that the invariant description is ultimately specified as a set of processable facts and axioms.

Key Classes:
(Object Class Invarient I,Object Class ID)
Owned Attribute Classes:

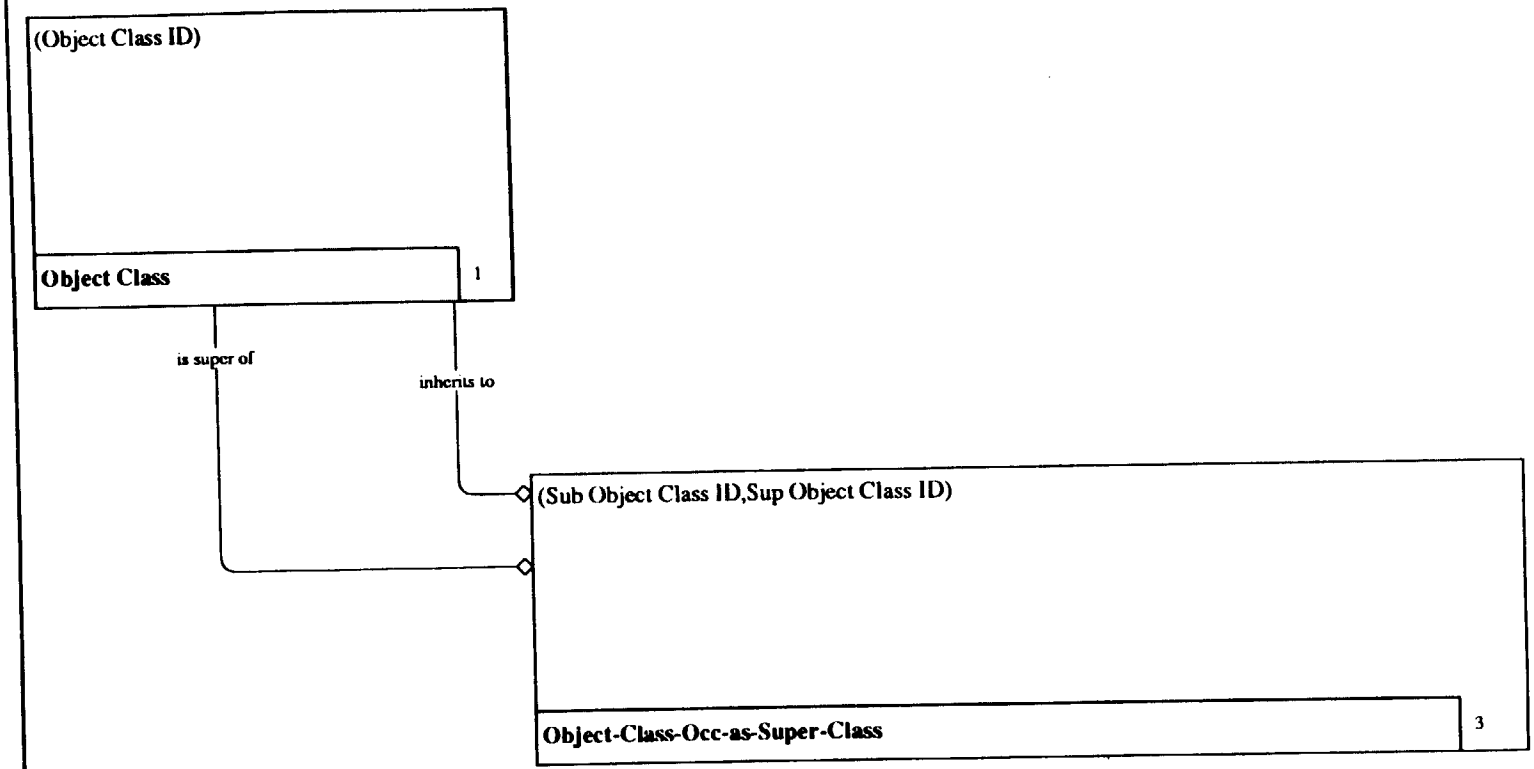
Name: Object Class Invarient G
Definition: This attribute contains a textual glossary description of an invarient.

Name: Object Class Invarient S
Definition: This attribute contains the specification of the invarient propositions associated with an object class. In the future these propositions will take the form of IISyCL specifications. For now they are simply structured text.

Name: Object Class Invarient I
Definition: This attribute records the unique id of an invarient associated with an object class.

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Object Class ID	Object Class	1	Object Class	1	respects

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



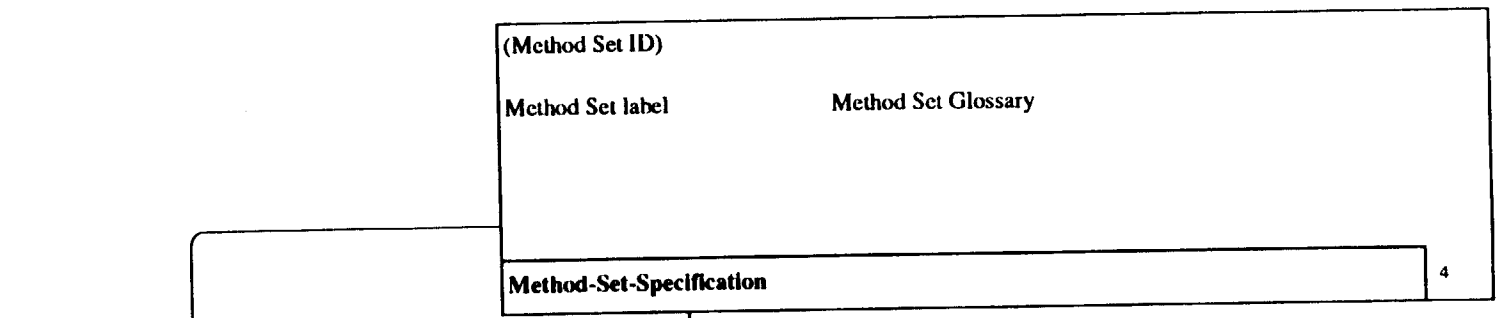
Node: E3	Title: Object-Class-Occ-as-Super-Class	Number: 21

Entity Class Definition: An Object Class can be the super class of 0, 1, or many other object classes.

Key Classes:
(Sub Object Class ID,Sup Object Class ID)

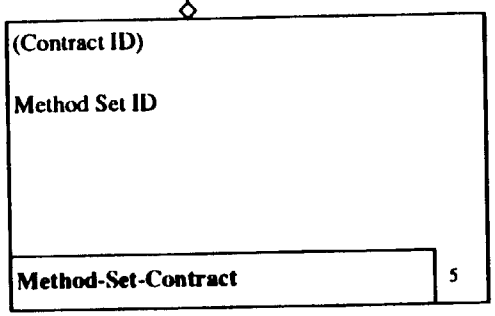
Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path		
			Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Sub Object Class ID	Object Class	1	Object Class	1	inherits from
Sup Object Class ID	Object Class	1	Object Class	1	has

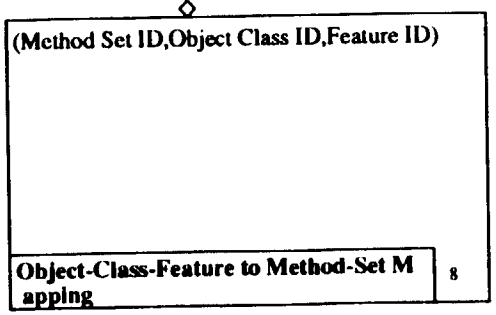
Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



has

is mapped to





Node: E4	Title: Method-Set-Specification	Number: 19

Entity Class Definition: A method set is an abstract characterization of a class of behaviors that is specialized as a routine in a particular object class. A method set may have a taxonomic parent, it will also have a protocol specification.

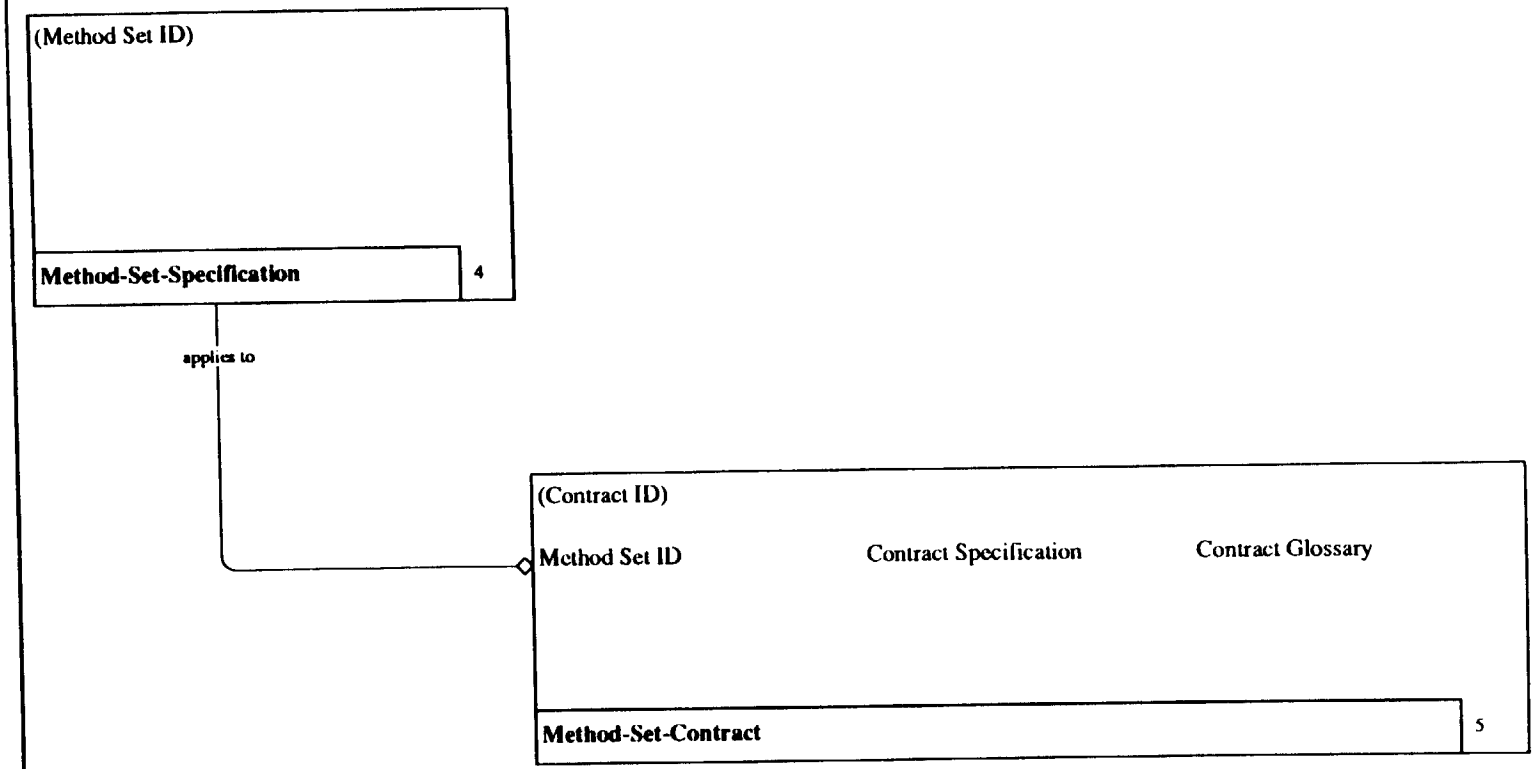
Key Classes:
(Method Set ID)
Owned Attribute Classes:

Name: Method Set Glossary
Definition: This attribute captures a textual description of a method set behavioral description/contract.

Name: Method Set ID
Definition: The unique identifier of a method set specification.

Name: Method Set label
Definition: The name of a method set that is displayed either in a method taxonomy diagram or a type diagram.

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



Node: E5	Title: Method-Set-Contract	Number: 17

Entity Class Definition: This entity class tracks the specification that a method set contracts to. It currently contains text which captures the content of method set contract data sheet.

Key Classes:
(Contract ID)
Owned Attribute Classes:

Name: Contract Glossary
Definition: This attribute captures an informal textual description of a contract specification.

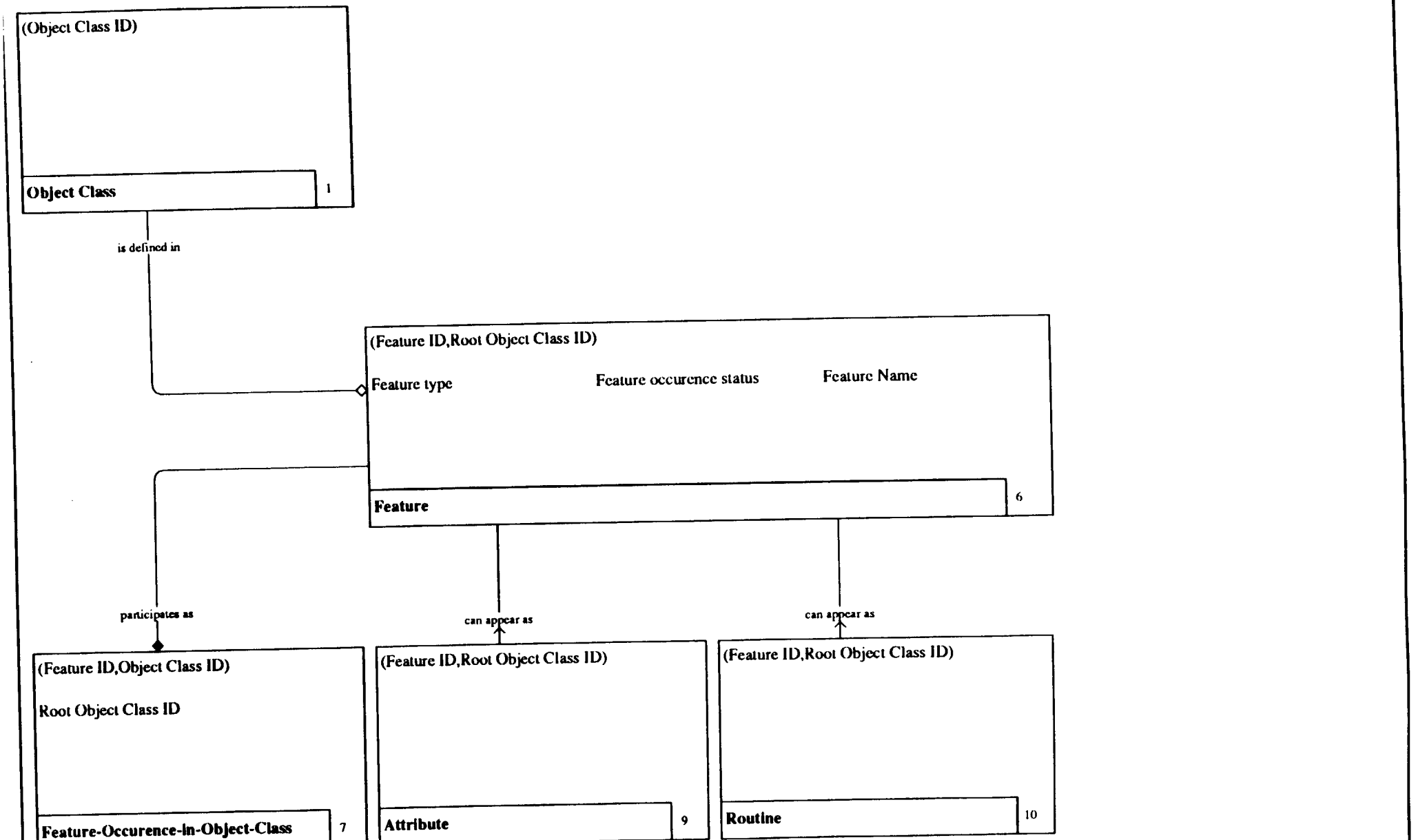
Name: Contract ID
Definition: The unique identifier of a behavior contract which can be used as a method set specification.

Name: Contract Specification
Definition: This attribute captures the specification of the behavior of a particular contract. In the future it is anticipated that such a specification will be posed in the form of IISyCL language statements. For this project it will take the form of structed text or pseudo code.

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Method Set ID	Method-Set-Specification	4	Method-Set-Specification	4	has

Node: E5 | Title: Glossary | Number:

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



Node: E6	Title: Feature	Number: 15

Entity Class Definition: The basic property or attribute concept used to associate both state and behavior to an object class. Features come in two basic flavors; routines and attributes. Each of these subtypes have two specializations (see the feature-type attribute class).

Key Classes:
(Feature ID,Root Object Class ID)
Owned Attribute Classes:

Name: Feature ID
Definition: The unique identifier of a feature.

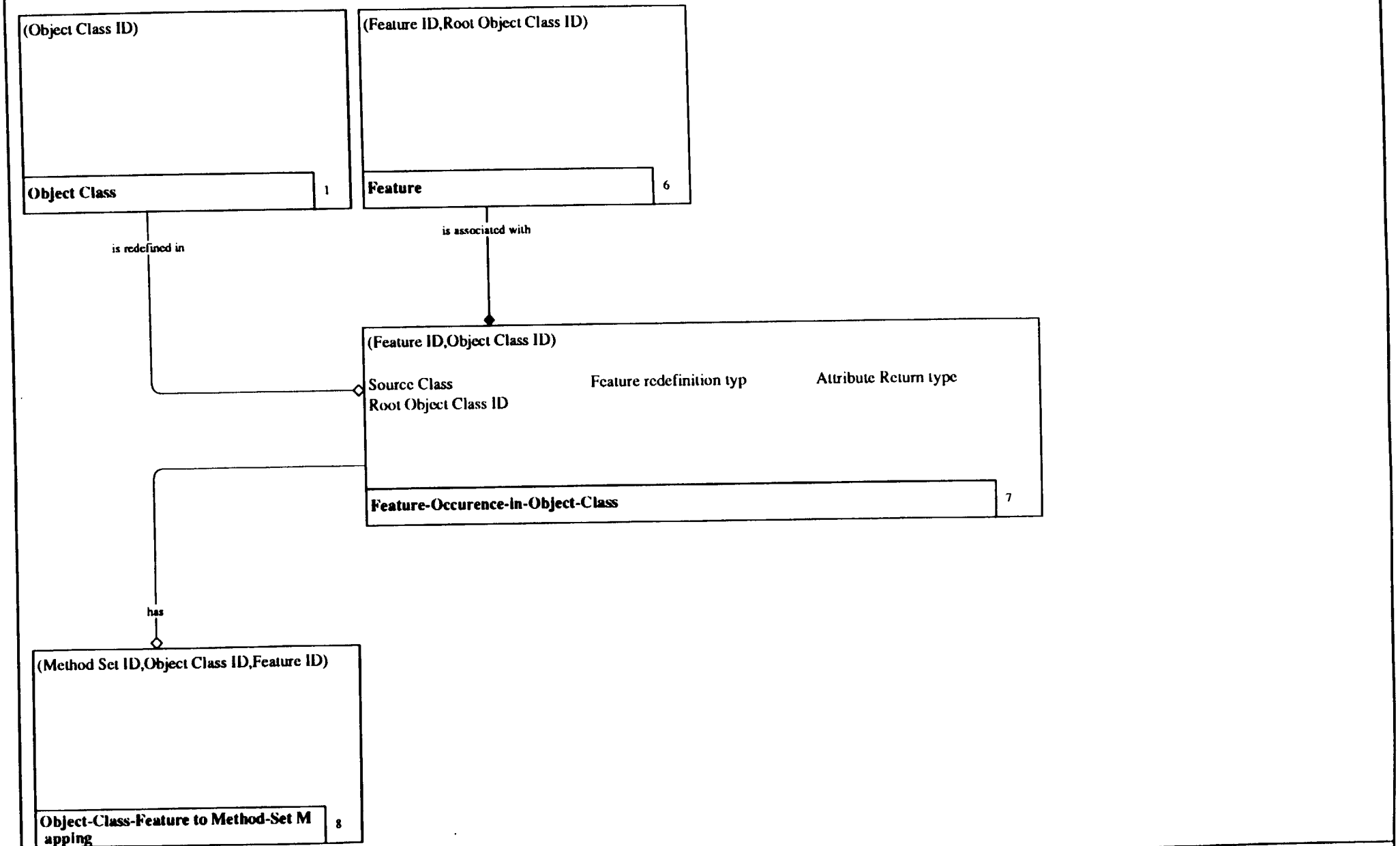
Name: Feature Name
Definition: The unique name of a feature.

Name: Feature occurence status
Definition: This attribute records for a feature occurence in an object class whether or not that feature is exported (public) or private (not exported from that class).

Name: Feature type
Definition: There are two basic feature types (attributes and routines).

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Root Object Class ID	Object Class	1	Object Class	1	is root of

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



Node: E7	Title: Feature-Occurence-in-Object-Class	Number: 13

Entity Class Definition: A feature can be directly present in many different object classes. Directly present implies that the feature is not defined in that class but rather is redefined in some way in that class.

Key Classes:
(Feature ID,Object Class ID)
Owned Attribute Classes:

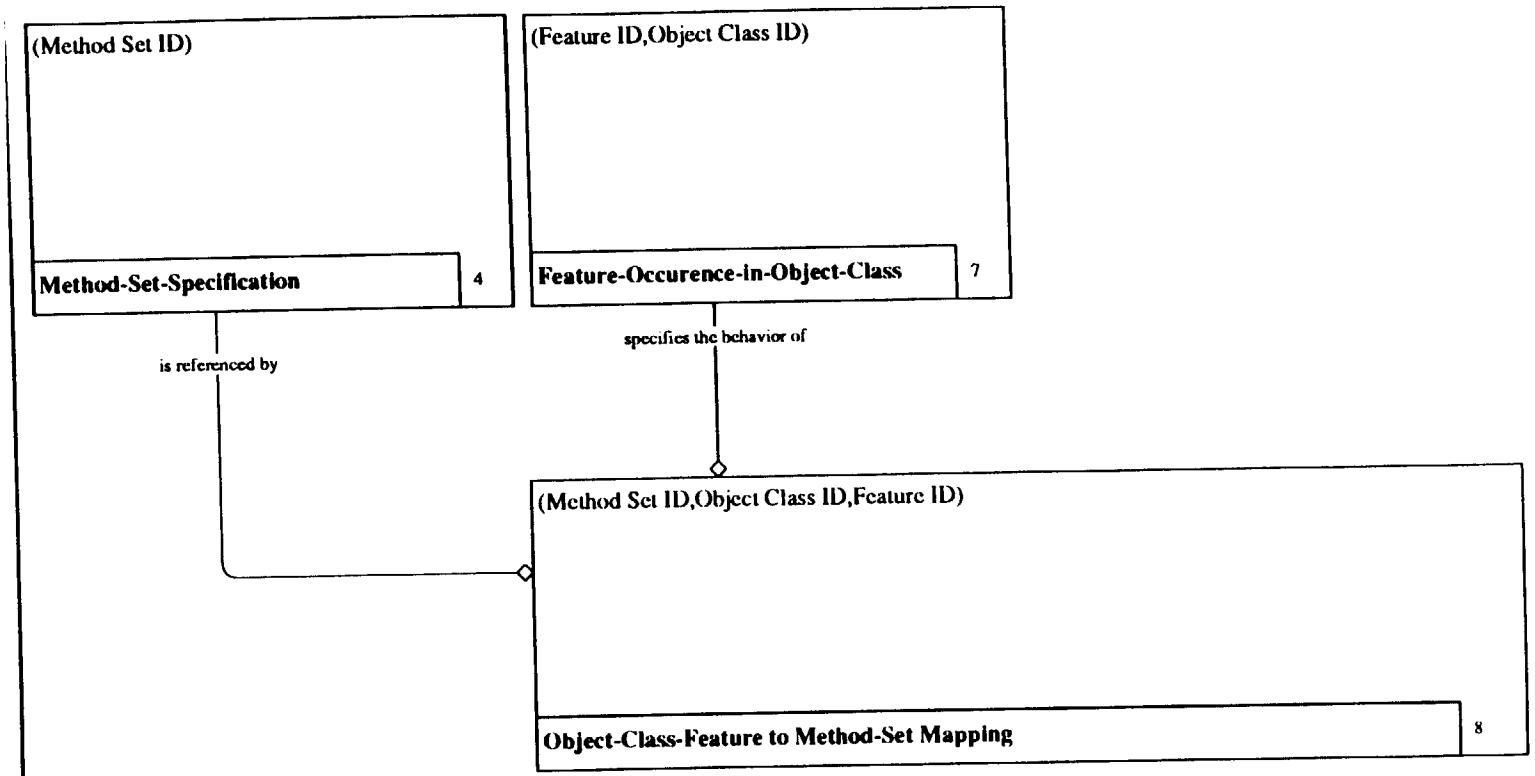
Name: Attribute Return type
Definition: All attributes have a return type, that being the type of the value returned by that attribute.

Name: Feature redefinition typ
Definition: This attribute can be one of (shown with symbol designator): 1) & additional contract (applicable to only routine type features 2) ! new taxonomic specification 3) + new method set 4) ^ now public 5) * now private

Name: Source Class
Definition: The nearest parent of a class in which a feature in the child class is directly present in that parent class.

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Root Object Class ID	Object Class	1	Feature	6	participates as
Feature ID	Feature	6	Feature	6	participates as
Object Class ID	Object Class	1	Object Class	1	displays

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



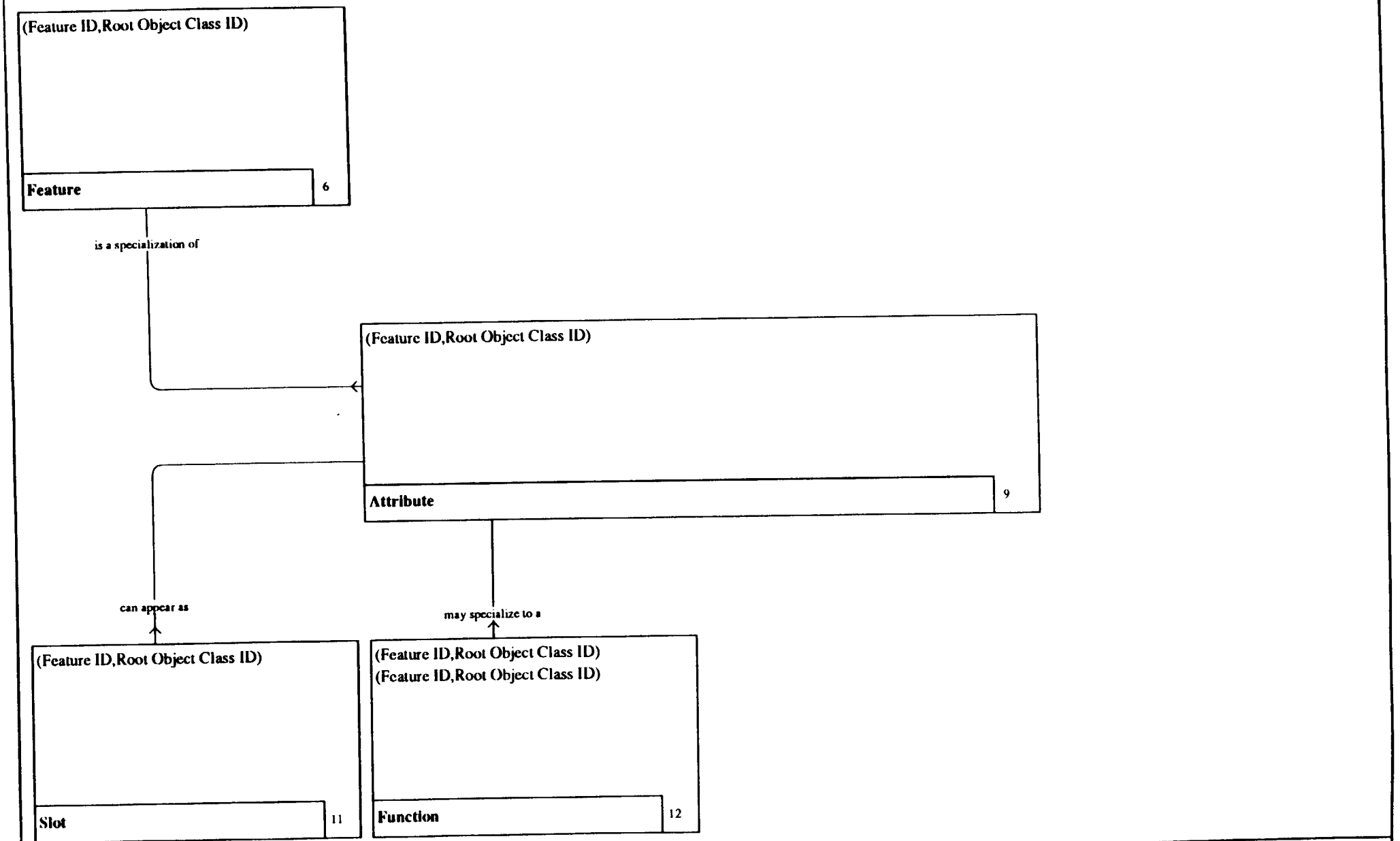
Node: E8	Title: Object-Class-Feature to Method-Set Mapping	Number: 11

Entity Class Definition: The recording of the mapping between object class feature and the method or methods which impliment that feature. This entity class records for a feature occurence within an object class the corresponding method set(s) describing the actual behavior contract of that feature.

Key Classes:
(Method Set ID,Object Class ID,Feature ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Method Set ID	Method-Set-Specification	4	Method-Set-Specification	4	is mapped to
Object Class ID	Object Class	1	Feature-Occurence-in-Object-Class	7	has
Feature ID	Feature	6	Feature-Occurence-in-Object-Class	7	has

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



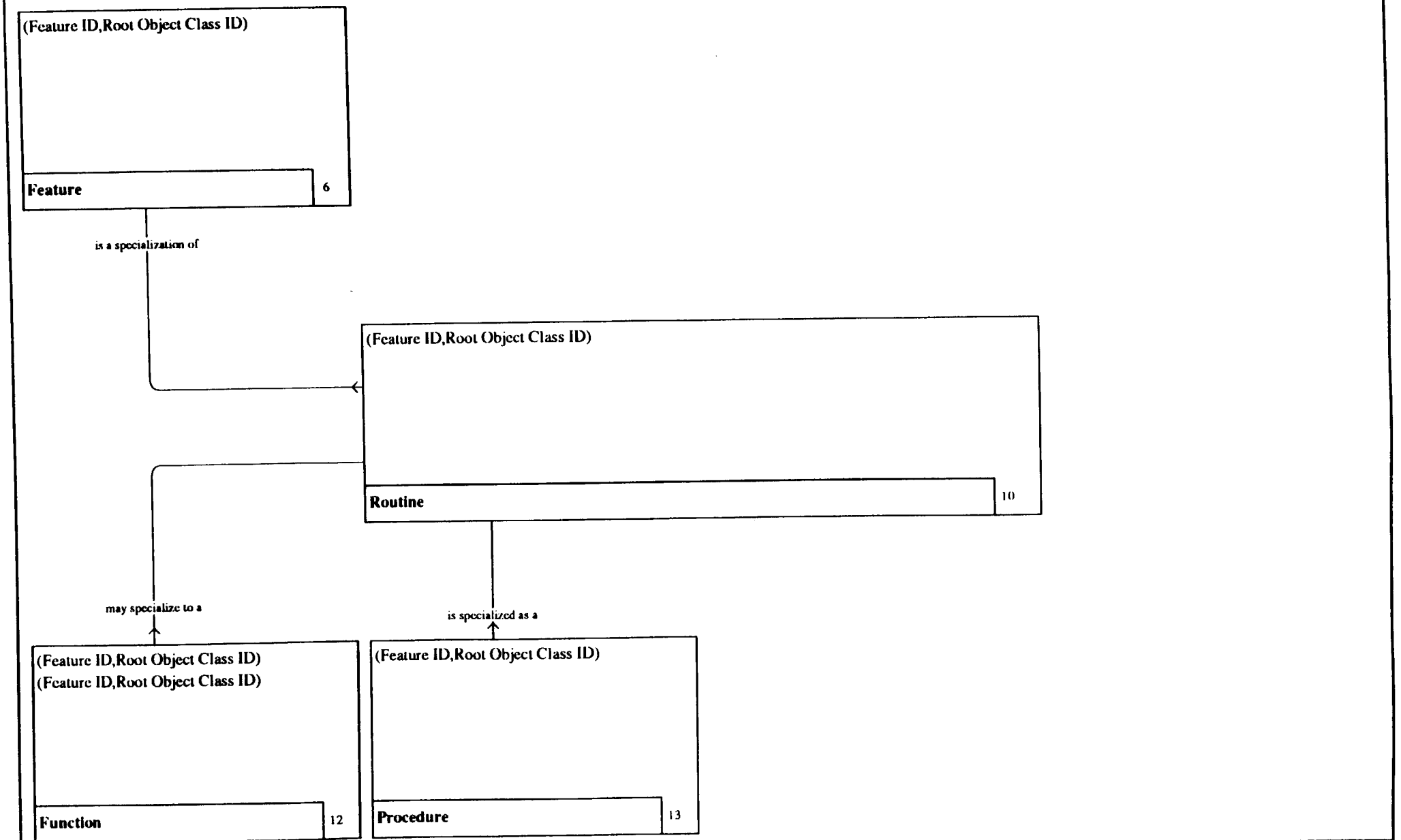
Node: E9	Title: Attribute	Number: 9

Entity Class Definition: A subtype of features that return values.

Key Classes:
(Feature ID,Root Object Class ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Root Object Class ID	Object Class	1	Feature	6	can appear as
Feature ID	Feature	6	Feature	6	can appear as

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE:	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



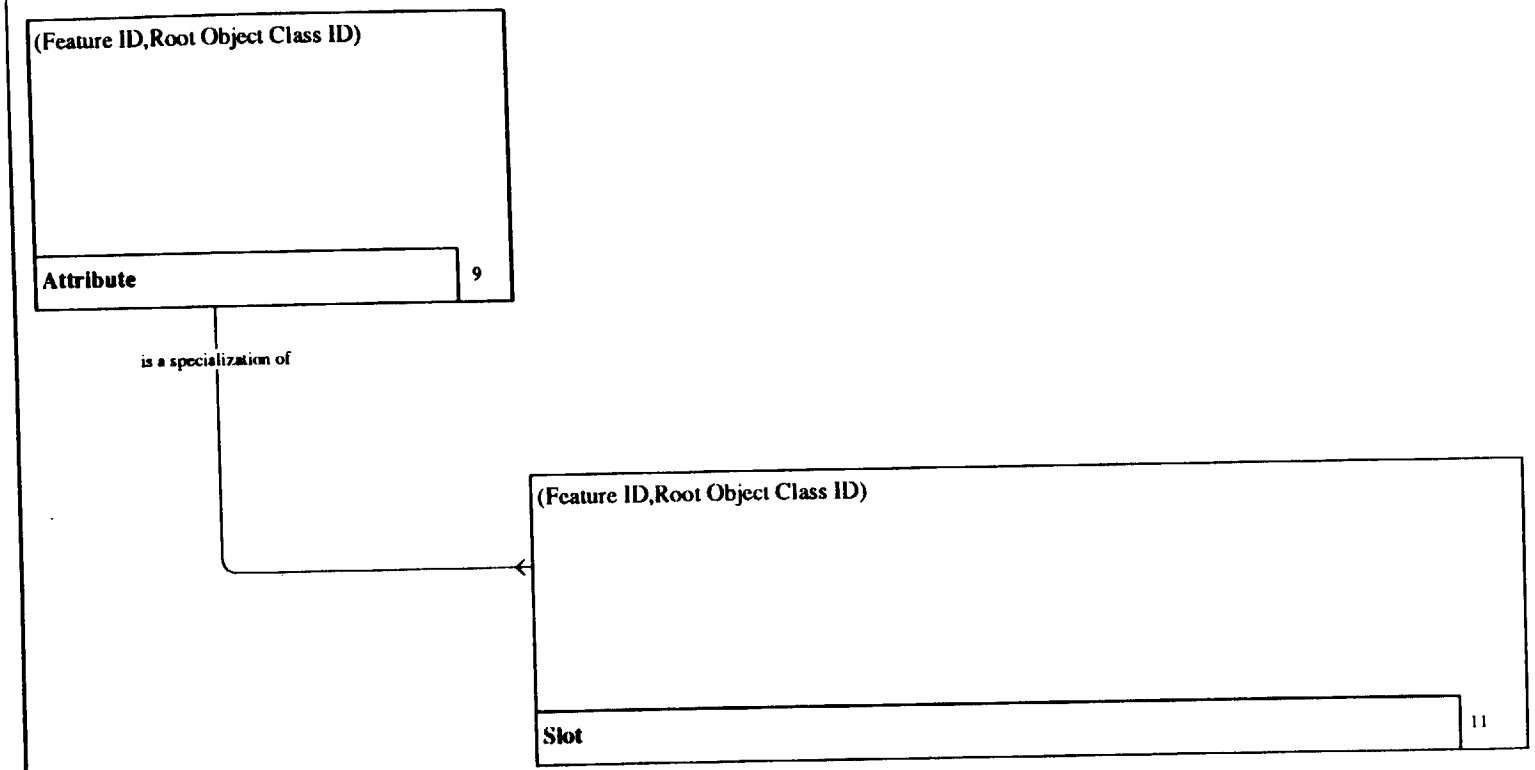
Node: E10	Title: Routine	Number: 7

Entity Class Definition: A subtype of features that can perform computation.

Key Classes:
(Feature ID,Root Object Class ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Root Object Class ID	Object Class	1	Feature	6	can appear as
Feature ID	Feature	6	Feature	6	can appear as

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE:	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



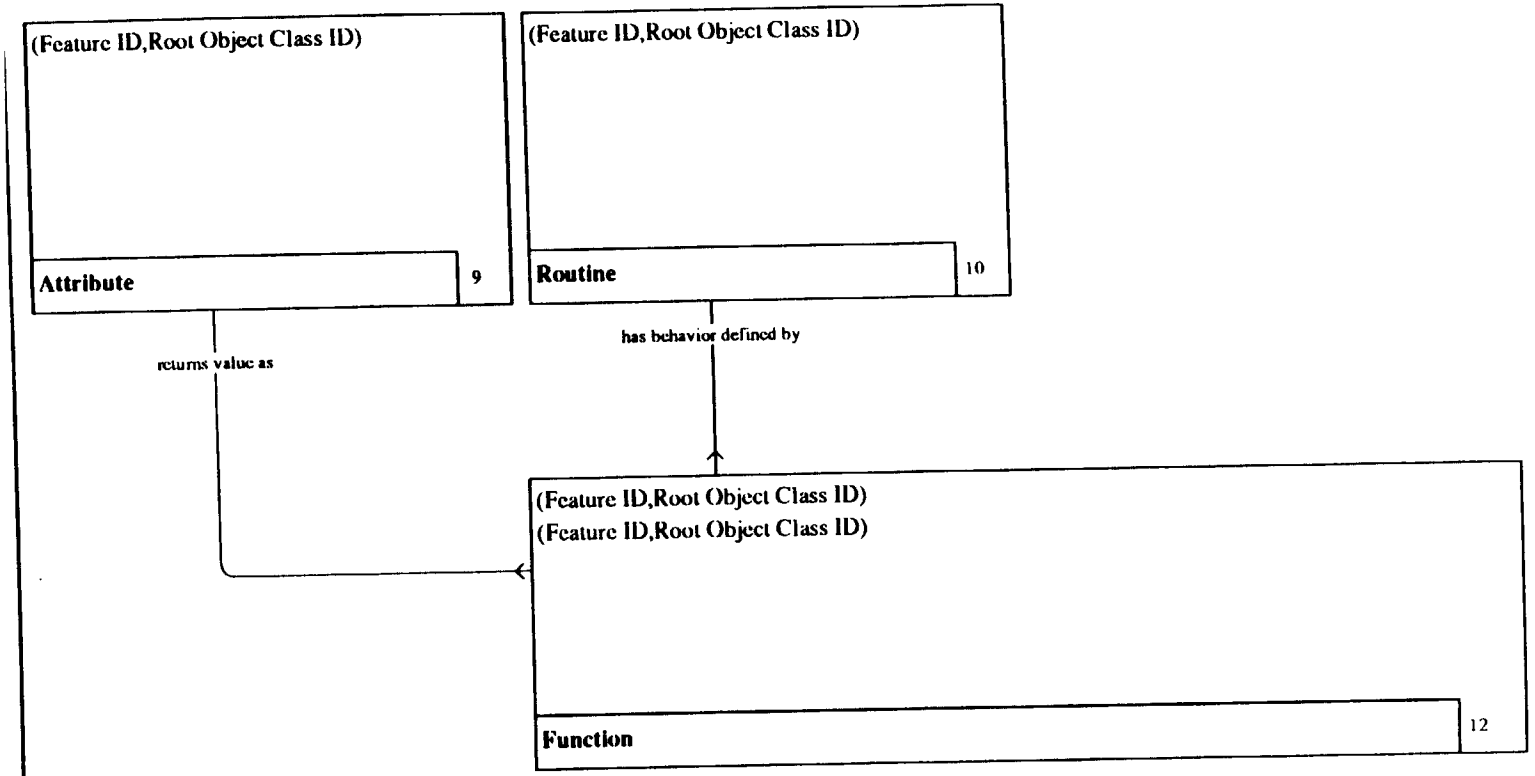
Node: E11	Title: Slot	Number: 5

Entity Class Definition: A subtype of attribute that only stores values.

Key Classes:
(Feature ID,Root Object Class ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path		
			Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Root Object Class ID	Object Class	1	Attribute	9	can appear as
Feature ID	Feature	6	Attribute	9	can appear as

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



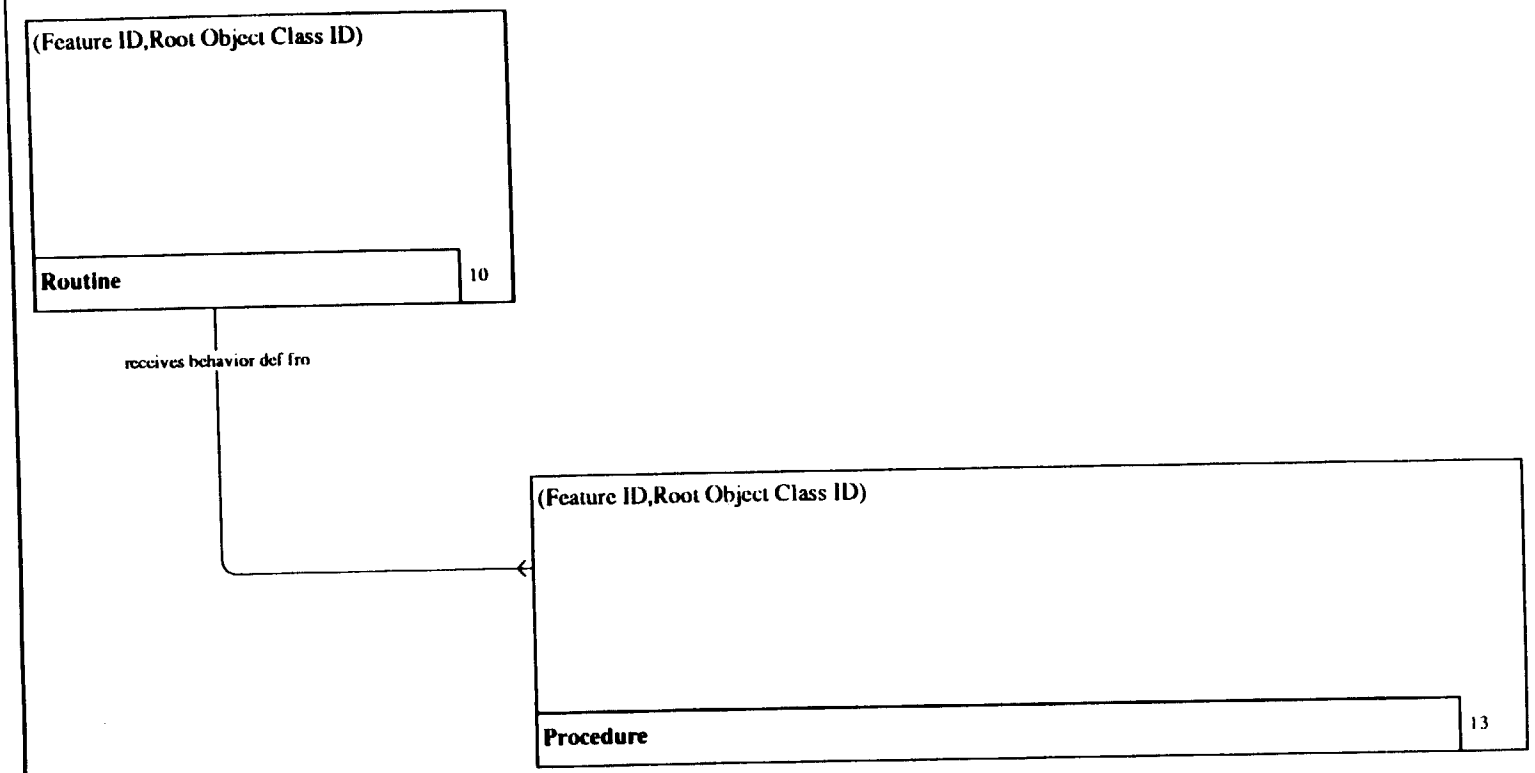
Node: E12	Title: Function	Number: 3

Entity Class Definition: A subtype of both attribute (it returns a value) and a subtype of routine (it performs calculation).

Key Classes:
(Feature ID,Root Object Class ID) (Feature ID,Root Object Class ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Root Object Class ID	Object Class	1	Routine	10	may specialize to a
Root Object Class ID	Object Class	1	Attribute	9	may specialize to a
Feature ID	Feature	6	Routine	10	may specialize to a
Feature ID	Feature	6	Attribute	9	may specialize to a

Used At:	Author: tom blinn	Date: 11-19-89	Working	READER	DATE	Context
	Project: IDEF4	Rev:	Draft			
	Notes: 1 2 3 4 5 6 7 8 9 10		Recommended			
			Publication			



Node: E13	Title: Procedure	Number:

Entity Class Definition: A subclass of routines that do not return values, they peform computations strictly for the side-effects.

Key Classes:
(Feature ID,Root Object Class ID)

Inherited Attribute Class(es)	Attribute Class Owned By: Entity Class Name	Number	Attribute Migration Path: Inherited From: Entity Class Name	Number	Inherited Through: Relation Class Name:
Root Object Class ID	Object Class	1	Routine	10	is specialized as a
Feature ID	Feature	6	Routine	10	is specialized as a

Copies of this publication have been deposited with the Texas State Library in compliance with the State Depository Law.