P-91

Automated IDEF3 and IDEF4 Systems Design Specification Document

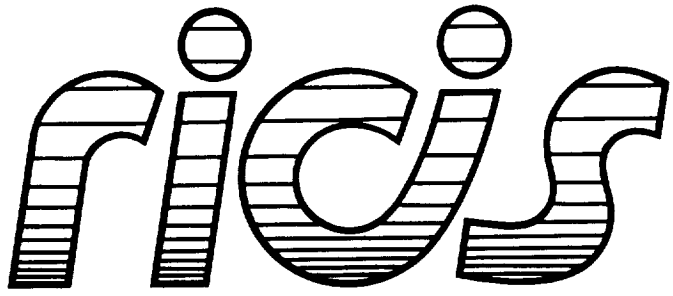
Patricia Griffith Friel
Thomas M. Blinn

Knowledge Based Systems, Inc.

November 20, 1989 - December 19, 1989

Cooperative Agreement NCC 9-16
Research Activity No. IM.15

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division



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

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

Automated IDEF3 and IDEF4 Systems Design Specification Document

Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Dr. Patricia Griffith Friel and 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 authors and should not be interpreted as representative of the official policies, either express or implied, of NASA or the United States Government.

Automated IDEF3 and IDEF4 Systems Design Specifications Document

An Interim Technical Report

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

Principal Investigators:
Dr. Patricia Griffith Friel
Thomas M. Blinn

Developed For: Software Technology Branch
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

November 20, 1989 - December 19, 1989

Table of Contents

List of Figures

1.0 Introduction

This document provides the initial design specification for the IDEF3 and IDEF4 systems that will provide automated support for IDEF3 and IDEF4 modeling. The IDEF3 method is a scenario-driven process flow description capture method intended to capture the knowledge about how a particular system works. The IDEF3 method provides modes to represent both (1) Process Flow Descriptions to capture the relationships between actions within the context of a specific scenario and (2) Object State Transition to capture the description of the allowable states and conditions for transition between those states of an object in the domain. The automated IDEF3 tool will provide support for both of these modes. The IDEF4 method is a graphically oriented methodology for the design of object oriented software systems.

Section 2.0 describes the philosophy of operation that was adopted for the design of these tools. It describes the general characteristics that the tool should have to provide effective support for these two methodologies.

Section 3.0 presents a mapping between the conceptual components of the two tools and the functional requirements specified in the functional requirements document, "IDEF3 and IDEF4 Automation System Requirements Document and System Environment Models. Each component of the two tools provide support for only a subset of the functional requirements. This section will indicate what functionality is provided by each component.

Finally, in Section 4.0, the detailed designs of the two tools are presented. Interestingly, these designs are presented using both IDEF3 and IDEF4. IDEF4 is used to define the various objects that will exist within the two tools while IDEF3 process descriptions define the procedures that would be performed (by the user and by the automated system in support of the user) to create IDEF3 descriptions and IDEF4 designs using the two tools. These process descriptions define the interaction between the user and the tools. Section 4.1 presents the design of the IDEF3 tool, including:

(1) IDEF4 Type Diagrams defining the basic data structures (classes) of the automated IDEF3 tool as well as the valid types that features of these classes may take. These classes would include the basic

IDEF3 objects (UOBs, Elaborations, etc...) as well as the organizational structures (Scenarios, Decompositions, etc...).

(2) IDEF4 Class Inheritance Diagrams to indicate the relationships between the classes defined for the IDEF3 tool.

(3) IDEF3 Process Descriptions to describe the procedures that the user and tool would perform to create a valid IDEF3 process description.

Section 4.2 presents the design of the IDEF4 tool, including:

(1) IDEF4 Type Diagrams, defining the basic data structures (classes) of the automated IDEF4 tool as well as the valid types that features of these classes may take. These classes would include the basic IDEF4 objects (Class, Feature, Method Set, etc...) as well as the organization structures (Diagrams).

(2) IDEF4 Class Inheritance Diagrams to indicate the relationships between the classes defined for the IDEF4 tool.

(3) IDEF3 Process Descriptions to describe the procedures that the user and tool would perform to create a valid IDEF4 system design.

2.0 Philosophy of Operation

One of the primary mechanisms used for descriptions of the world is relating a story in terms of an ordered sequence of events or activities. The original IDEFs were developed for the purpose of enhancing communication among people who needed to decide how their existing systems were to be integrated. IDEF0 was designed to allow a graceful expansion of the description of a systems' functions through the process of function decomposition and categorization of the relations between functions (i.e., in terms of the Input, Output, Control, and Mechanism classification). IDEF1 was designed to allow the description of the information that an organization deems important to manage in order to accomplish its objectives. The third IDEF (IDEF2) was originally intended as a user interface modeling method. However, since the ICAM Program needed a simulation modeling tool, the resulting IDEF2 was a method for representing the time varying behavior of resources in a manufacturing system, providing a framework for specification of math model based simulations. It was the intent of the methodology program within ICAM to rectify this situation but limitation of funding did not allow this to happen. As a result, the lack of a method which would support the structuring of descriptions of the user view of a system has been a major shortcoming of the IDEF system. The basic problem from a methodology point of view is the need to distinguish between a description of what a system (existing or proposed) is supposed to do and a representative simulation model that will predict what a system will do. The latter was the focus of IDEF2, the former is the focus of IDEF3.

The development of IDEF4 came from the recognition that the modularity, maintainability and code reusability that results from the object oriented programming paradigm can be realized in traditional data processing applications. The proven ability of the object oriented programming paradigm to support data level integration in large complex distributed systems is also a major factor in the widespread interest in this technology from the traditional data processing community. The object oriented programming paradigm provides the developer with an abstract view of his program as composed of a set of state maintaining objects which define the behavior of the program by the protocol of their interactions. An object consists of a set of local state defining attributes and a set of methods (procedures) that define the behavior of that particular object and its relationship to the other objects that

make up the system. IDEF4 was developed as a design tool for software designers who use object-oriented languages such as the Common LISP Object System, Flavors, C++, SmallTalk, Objective C and others. Since effective usage of the object-oriented paradigm requires a different thought process than used with conventional procedural or database languages, standard methodologies such as structure charts, data flow diagrams, and traditional data design models (hierarchical, relational, and network) are not sufficient. IDEF4 seeks to provide the necessary facilities to support the object-oriented design decision making process.

A logical step after the development of a methodology is the automation of that methodology. As with other processes, automating IDEF3 and IDEF4 should reduce the turnaround time required to produce the descriptions and designs as well as increase the productivity of the domain experts. But, while automation addresses the needs of the domain experts, initial prototyping can serve other purposes. In this light, the prototype IDEF3 and IDEF4 systems will attempt to address several other areas.

First of all, the prototype IDEF3 and IDEF4 systems will demonstrate the viability of the two methodologies. Since IDEF3 and IDEF4 are very young methodologies, the prototype tools will serve as a platform for demonstrating the concepts and intentions of the methodologies. Additionally, the prototypes will showcase new technology and demonstrate the effect the technology can have on automated tools. Perhaps the most significant technology to be incorporated in the prototypes will be an object oriented database system. The object oriented database technology is still evolving and very few commercial products exist. By using an object oriented system, the prototypes will demonstrate the importance of this new technology.

While the methodologies and the technology used to develop the prototypes are important, a more basic need to be satisfied by the prototype development is the demonstration of the basic functionality required to automate the methodology. In doing this, it is possible to identify those components of the methodologies that prove most difficult to automate. With this knowledge, a more accurate development strategy can be produced when attempting to build the full scale automated tools. Potential areas where additional work will be required after the initial prototyping deal with the integration issues between various modeling methodologies and the

database issues revolving around the object oriented database architecture.

In addition to these functional goals, the IDEF3 and IDEF4 prototypes should display the following operational characteristics:

Ease of Data Entry

The system should be simple to use. The tool would not be very useful if the operation of the tool made the description and design processes more complicated than it would be without an automated tool.

No Redundant Data Entry

The system should not require the user to input information more that once. Once a datum exists within the system, the datum should be accessible for use at any time.

Integratability

The tools should have hooks for integration with other tools (particularly IDEFØ and IDEF1 tools) and provide means of using process descriptions and designs for automated analysis (i.e., logical consistency, qualitative simulation and code generation).

Presentation/Graphic Based Operation

The most obvious characteristic will be the systems user interface. The design of the two tools has proceeded with the assumption that the user interface will be a presentation/graphic based operation. What this means is that the interface will be frame (window) based with displayed objects being mouse sensitive. The objects on the screen will be easily manipulated using the mouse as an input device.

Multi-mode Input and Editing Capability

An additional characteristic that the interface should demonstrate is a multi-mode operational ability. The multi-mode input capability will give the user the greatest amount of freedom in developing their descriptions and designs. This is especially important since IDEF3 and IDEF4 are relatively new methodologies that do not have well defined strategies for model development. These

multi-modes will serve as experiments to determine which modes are useful and which modes are useless as well as to define an organized development strategy for process descriptions and object oriented designs.

Knowledge Based Operation

Probably the most important aspect of these tools is that they demonstrate a good deal of knowledge based operation. The tools should have an understanding of the effect that certain operations will have on the process description or design database. When these operations are performed, the tool should have the ability to revise the model database intelligently so that the number of operations required by the user is reduced to a minimum. Also, the tool should have some conflict resolution strategies encoded into the system to ensure that the models are consistent at all times. The user should not be required to resolve conflicts that can be safely resolved using algorithmic or heuristic methods.

By adhering to these concepts, the prototype tools will give the expert the support necessary to effectively develop IDEF3 process flow descriptions and IDEF4 object oriented designs. The IDEF3 tool will allow the expert to:

- Develop and define IDEF3 process description
- Evolve an existing IDEF3 process description
- Identify valid states that objects may exist in during a process
- Expand on information represented in IDEFØ and IDEF1 models

The IDEF4 tool will allow the user to:

- Develop and define IDEF4 design representation
- Evolve an existing IDEF4 design
- Modify the design of an existing software system
- Use an IDEF4 design to maintain an Object Oriented Software System

To provide these capabilities and to produce designs that could be beneficial to the design of full-scale production tools, several design goals were identified that regulated the development of the prototype tool designs. These goals are to:

- Design tools that demonstrate a high degree of functionality
- Design tools that incorporate knowledge based characteristics
- Design tools that can be upgraded to fully functional tools (on Symbolics)
- Design tools that will adapt to other hardware platforms
- Design tools that demonstrate the advantages of an object oriented database system
- Design tools that can be integrated with both analysis and requirements tools and with code generation tools in programming environments.

IDEF3 and IDEF4 are powerful description capture and design methodologies. The development of prototype automated tools for these methods provides a showcase for the capabilities of the methods as well as the opportunity to demonstrate the effect that new technologies can have on the development of software systems.

To effectively automate the methodologies requires that the tools exhibit certain functional qualities such as knowledge based operations, multi-mode editing capability, an no redundant data entry, among others. In addition to this, the tools must provide the user with a robust set of capabilities that will allow them to easily model and design their systems. It is for these reasons that the design goals enumerated above have strongly influenced the design of the prototype IDEF3 and IDEF4 tools.

3.0 Functional Summary

The focus of this document now shifts to the actual designs of the IDEF3 and IDEF4 tools. This section describes the components that will make up the two tools and what functionality each component will provide. As such, this section will provide a mapping between the components of the two tools and the functional requirements defined for the two tools in "IDEF3 and IDEF4 Automation System Requirements Document and System Environment Model".

3.1 IDEF3 Functional Operation

Conceptually, the IDEF3 tool will consist of several interacting utilities:

Unit of Behavior Pool Browser
Object State Browser
IDEF3 Process Flow Diagram Facility
Object State Transition Diagram Facility

The Unit of Behavior Pool Browser will provide a listing of all the Units of Behavior that have been defined within the current scenario. The browser will provide the ability to find what process descriptions make use of a specific UOB, obtain information on the UOB's elaboration and decompositions, as well as modify the UOB.

The Object State Browser will provide a listing of all Object States that have been defined within the current scenario. This component will support the selection, modification, and viewing of the various object states.

The Process Flow Diagram Facility allows the UOBs, Links, and Junctions to be laid out into a valid process flow description. This mode will allow movement between different descriptions that are part of the same scenario as well as view the decompositions of UOBs that make up a process flow description.

The Object State Transition Diagram Facility provides the capability necessary to define a valid Object State Transition network. It will allow for the addition of object states to the diagram, the specification of transition arcs between objects states, and the labelling of transitions arcs with other state transition diagrams.

Figure 1 provides a mapping between these components and the functional requirements defined in the requirements document. The sections mentions in the diagram refer to sections in the IDEF3 and IDEF4 Automation System Requirements Document. Notice that a certain degree of overlap exists between the various modes. This characteristic gives the domain expert the greatest degree of freedom in developing the process descriptions and should allow for the rapid development of those process descriptions.

	IDEF3 Process Flow Diagram Facility	Object State Transition Diagram Facility	Unit of Behavior Pool Browser	Object State Browser
Unit of Behavior Operations (Section 3.1.1.1)	√		√	
Link Operations (Section 3.1.1.2)	√			
Junction Operations (Section 3.1.1.3)	√			
Reference Operations (Section 3.1.1.4)	√	√		
Elaboration Operations (Section 3.1.1.5)	√	√	√	√
Object State Operations (Section 3.1.2.1)		√		√
Transition Arc Operations (Section 3.1.2.2)		√		
Scenario Operations (Section 3.2.1)	√			
Decomposition Operations (Section 3.2.2)	√		√	
Information Management Operations (Section 3.5)	√	√	√	√

Figure 1. IDEF3 Functional Matrix

3.2 IDEF4 Functional Operation

Conceptually, the IDEF4 tool will consist of several interacting modes of operation as well:

Class Submodel Browser
Feature Pool Browser
Method Submodel Browser
IDEF4 Diagram Facility

The Class Submodel Browser will provide a listing of all classes that have been defined in the current design. With each class, the user is able to access and modify the information associated with a class, such as its features and its related parent or child classes. Changes to a class' relationship with other classes will be automatically reflected in any diagrams in which the class occurs.

The Feature Pool Browser will provide a listing of all features that have been defined in the current design. The features will be defined as separate objects in this system to allow for the situation where a feature will be defined, but the class to which the feature will belong has not been decided. Accordingly, within the Feature Pool Browser, the designer will have the ability to assign a feature to a particular class in addition to the normal feature operations such as creation, editing, and deletion.

The Method Submodel Browser will provide a listing of all method sets that have been defined in the current design. In this browser, it will be possible to view information associated with a method set, as well as modify that information. The most important operations to be supported within this mode is the creation, editing, and deletion of method sets, the specification of taxonomic links between method sets, and dispatch mapping between a method set and a feature of a class.

Finally, the IDEF4 Diagram Facility provides for the generation and manipulation of all the diagrams specified in an IDEF4 design. Also within this facility, the designer will have the ability to make changes to classes, features, and method sets that participate in the diagram. This will prevent unnecessary changes between the various object browsers and the diagram facility.

Figure 2 displays a mapping between these four units that will make up the IDEF4 tool and the functional requirements defined in the requirements document. The sections referred to in the diagram are sections in the IDEF3 and IDEF4 Automation System Requirements Document. Notice that there is a certain degree of overlap between the tool components and the operations that the components support. This characteristic will provide the user with the greatest degree of flexibility in defining and modifying their system design by limiting the number of mode changes required to make adjustments to the design.

	IDEF4 Diagram Facility	Class Submodel Browser	Feature Pool Browser	Method Submodel Browser
Class Operations (Section 4.1.1)	√	√		
Feature Operations (Section 4.1.2)	√	√	√	
Inheritance Link Operations (Section 4.1.3)	√	√		
Type Link Operations (Section 4.1.4)	√	√	√	
Method Set Operations (Section 4.1.5)	√			√
Class Inheritance Diagram Operations (Section 4.2.1)	√			
Type Diagram Operations (Section 4.2.2)	√			
Protocol Diagram Operations (Section 4.2.3)	√			
Method Taxonomy Diagram Operations (Section 4.2.4)	√			
Client Diagram Operations (Section 4.2.5)	√			
Customized Diagram Operations (Section 4.2.6)	√			
Information Management Operations (Section 4.5)	√	√	√	√

Figure 2. IDEF4 Functional Matrix

4.0 Automated IDEF3 and IDEF4 Tool Design

This section provides a detailed description of the current design for the Automated IDEF3 and IDEF4 Tools. The section will make use of IDEF3 and IDEF4 descriptions to relay the tool design as well as textual discussion to elaborate on the design and to explain why certain design choices were made. IDEF4 diagrams will define the data structures (objects) that will exist within the IDEF3 and IDEF4 tools while IDEF3 process flow descriptions will be used to relay the interactions a person would have with the tools while developing their process descriptions or object oriented designs.

4.1 IDEF3 Design Components

Up until now, the discussion has been focused on both the IDEF3 and IDEF4 tools. At this time, a more detailed analysis of the IDEF3 design will be presented while the IDEF4 design will be presented in Section 4.2. This discussion begins with the definition of the classes that will exist within the IDEF3 system along with their inheritance relationships to each other.

4.1.1 IDEF3 Data Structures

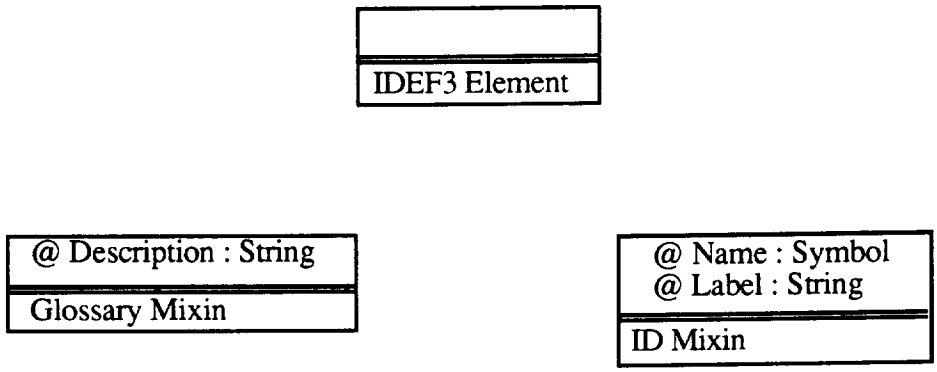


Figure 3. IDEF3 Mixin Classes Type Diagram

Figure 3 displays the type diagram of three classes that will be used for 'mixin' purposes. What this means is that instances of these classes will not exist within the IDEF3 system. Instead, the characteristics of these classes will be 'mixed' into other classes through feature inheritance. These classes represent a group of features and functionality that are common to several classes in the

system. As a result, these common elements are pulled out into their own classes so that the code to implement the operations will be developed only once.

4.1.1.1 IDEF3 Element

This class represents the basic functionality required for the manipulation, display, and presentation of any element in the IDEF3 system. As yet, no features have been identified, but various display operations have been defined for the IDEF3 elements.

4.1.1.2 Glossary Mixin

This class represents a textual description that will be attached to instances of classes that inherit from this class. The only feature defined for Glossary Mixin is Description, a slot accepting a value of type String. This description captures information explaining the purpose for the object to which the description is attached.

4.1.1.3 ID Mixin

This mixin class will be used for identification purposes. Its two attributes, Name and Label will be used to uniquely identify an object. The Name slot accepts values of type Symbol while the the Label slot accepts values of type String. In cases where only the name is required to uniquely identify an object, the label attribute will take a nil value.

Figure 4 contains the type diagram for the objects that make up the process flow descriptions. This diagram describes the individual objects that make up the process descriptions as well as the organizational objects that structure the description.

4.1.1.4 Linkable Object

This class is really a mixin class but was included in this diagram to indicate its features' return types. The purpose of this class is to indicate that any instance of a class that mixes in Linkable Object can be linked to other objects. As such, a linkable object can have a link coming into the object (In Link) and a link leaving the object (Out Link). These attributes return values of type Link.

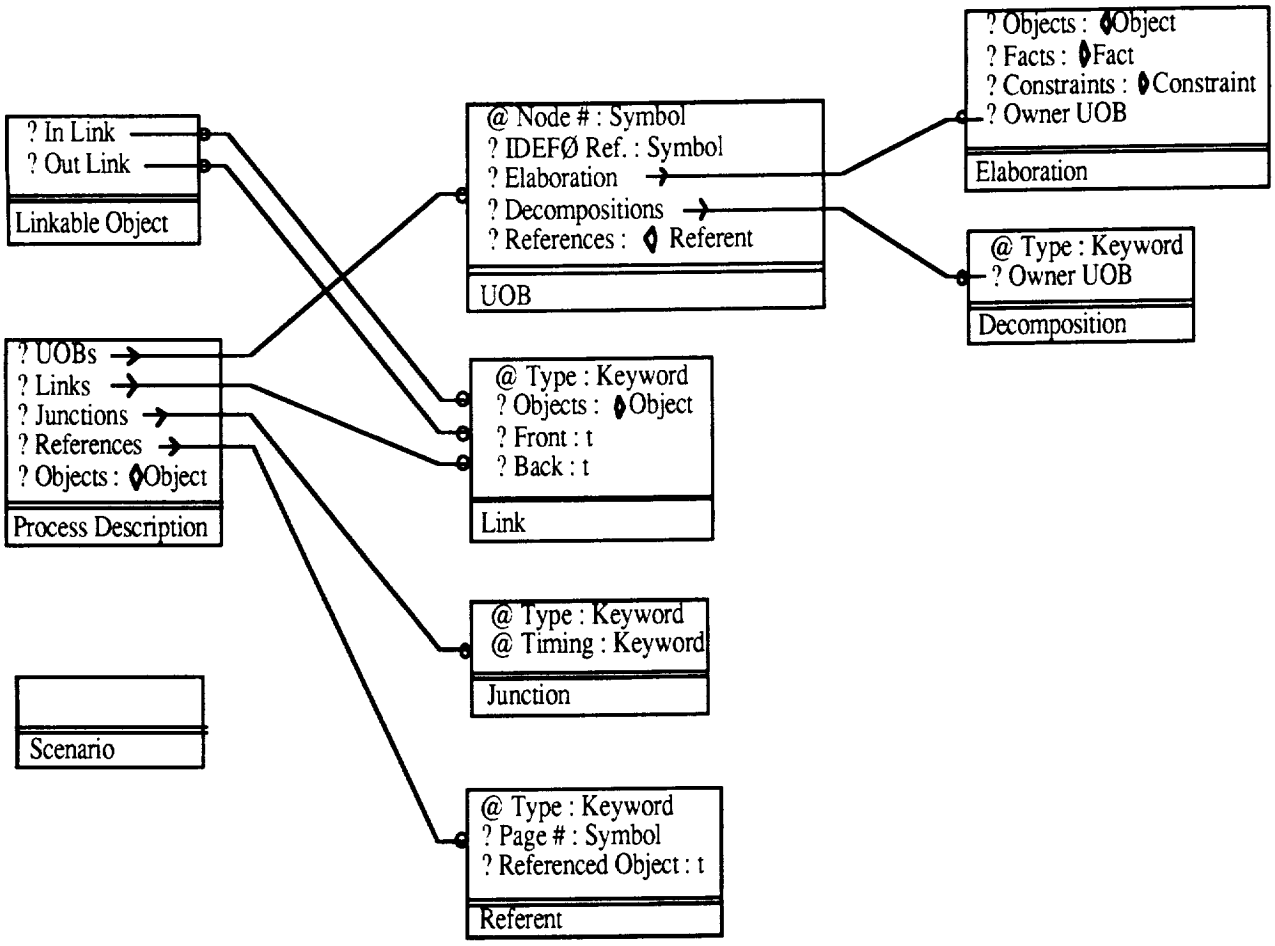


Figure 4. Process Flow Objects Type Diagram

4.1.1.5 Scenario

The Scenario class is an organizing structure that represents the highest level of organization in an IDEF3 description. An IDEF3 process flow description consists of several levels of process descriptions, where a UOB can be described in greater detail by attaching decompositions to that UOB. However, the top level process description is not a decomposition. Instead, this top level is a scenario. The scenario class does not have any owned features because the scenario is just a special case of a decomposition. The scenario's functionality will be derived through inheritance relationships to be discussed later in this section.

4.1.1.6 Process Description

The process description is the basic organizing unit in a process flow description. This class will maintain all the

objects that are part of a certain process flow description. UOBs is an attribute that will access the list of UOBs that are part of the process description. The attribute Links will maintain a list of links that relate the UOBs, Junctions, and Referents in a process description. Junctions is an attribute that will maintain a list of all Junctions in the description. Referents is an attribute that will maintain a list of Referents that occur in the process description. Finally, Objects will be a list of Objects that are somehow manipulated within this process description.

4.1.1.7 Decomposition

The Decomposition class represents an object that is associated with a Unit of Behavior. A decomposition gives a more detailed description of its parent UOB. A decomposition can be one of two types: objective or view. The Type attribute will indicate what type a particular decomposition is. The Decomposition class must also be attached to a Unit of Behavior. The Owner UOB attribute will provide a link to the Unit of Behavior to which the decomposition is attached.

4.1.1.8 UOB

The UOB class represents the Unit of Behavior entity in IDEF3. Every UOB has an identifying node number that is a Symbol. There is also an optional IDEFØ Ref. attribute. This Symbol, if provided, will reference an IDEFØ activity that the UOB is somehow associated with. A UOBs Elaboration will be referenced through the Elaboration attribute and the UOBs decompositions will be referenced through a list of decompositions maintained by the Decompositions slot. Finally, the UOB class also has a Referents attribute. This feature was added mainly to make the implementation of some functionality easier. Its purpose is to keep a list of all Referents that make use of this UOB. This will make searching for references to this UOB much easier.

4.1.1.8 Elaboration

The Elaboration class represents the IDEF3 Elaboration. The Owner UOB attribute takes a UOB as its value and is a link between an Elaboration and the UOB that the elaboration describes. The other three attributes maintain

the information that make up the elaboration. Objects keeps a list of objects that are somehow manipulated by the UOB. Facts keeps a list of facts that define the current state of the objects. Finally, Constraints maintains a list of constraints that the objects must satisfy.

4.1.1.9 Referent

The Referent class represents the IDEF3 Referent. A referent has three features: a keyword type, indicating the type of reference being made; a Page # symbol, representing the page number where the reference is described further; and a Referenced Object, accessing an object that is being referenced.

4.1.1.10 Junction

The Junction class provided for branching in the process descriptions. A junction has a Type, indicating And, Or, or XOR, and a Timing, indicating Asynchronous or Synchronous, attribute. Both of these attributes accept keyword values.

4.1.1.11 Link

The Link class defines the relations between linkable objects. A link can have a Type to specify whether it is a precedence, object flow, or relational link. If the link is an object flow link, then the objects that flow through the link are attached to the Objects attribute. Finally, the two linkable objects that are related by the link are specified by the Front and Back attributes.

Figure 5 presents the type diagram for the classes defined for the Object State Transition Diagrams in IDEF3. These diagrams define the various states that an object may exist in as well as the constraints that must be satisfied before a transition can take place.

4.1.1.12 Object

The Object class simply represents an object that play some role in a process description and is somehow affected by that process description. Its Elaborations attribute refers to any Elaborations in the process description that refer to this object while the Flows attribute references any Object Flow links that indicate this object is to flow between two Units of Behavior.

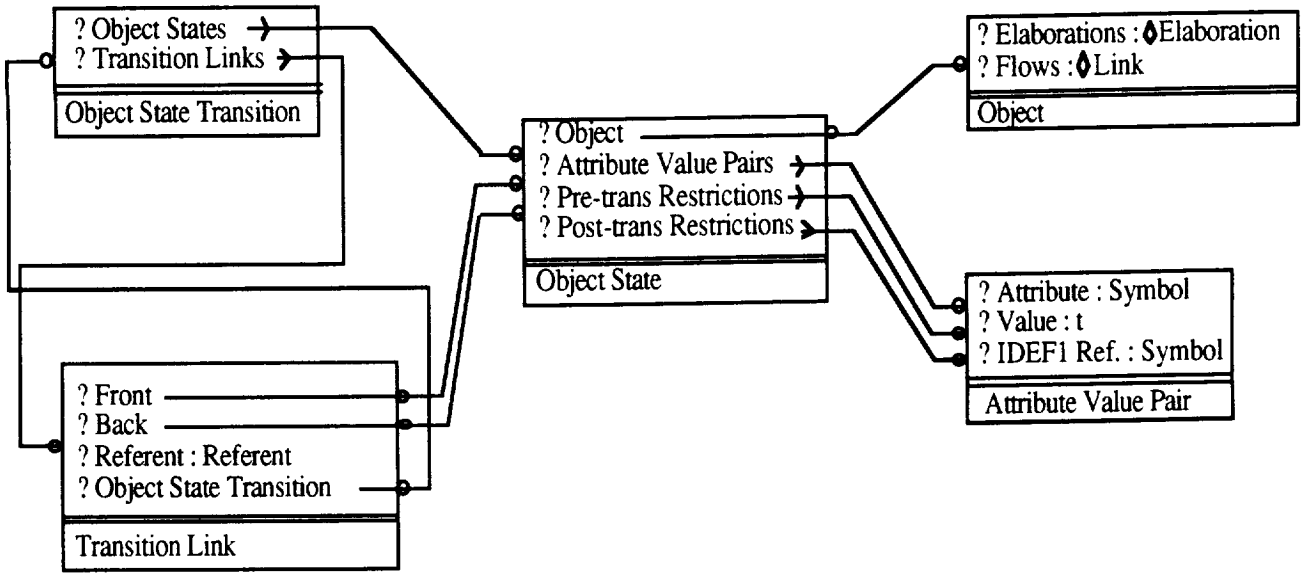


Figure 5. Object State Objects Type Diagram

4.1.1.13 Attribute Value Pair

The Attribute Value Pair class will be used to define object states and constraints on object. The pair will represent an object attribute and a value that the attribute must adhere to for the state to exist of the constraint to be satisfied. The Attribute feature refers to the attribute of the object., the Value feature indicates the value that the Attribute must take, and the IDEF1 Ref. feature indicates an IDEF1 Attribute Class with which the Attribute is associated.

4.1.1.14 Object State

The Object State class is used to define the state of an object. The Object attribute references the object for which the object state is being defined. The list of attribute value pairs maintained in the Attribute Value Pairs attribute actually defines the state while Pre-trans Restriction and Post-trans Restriction attributes maintain lists of constraint defining attribute value pairs that must be satisfied before beginning a transition and before completing a transition, respectively.

4.1.1.15 Transition Link

The Transition Link defines a transition between two object states. These two states are referred to by the Front and Back attributes of the link. Also, the link can have a referent attached to it through the Referent attribute to indicate a process flow that must be executed to effect the transition. Similarly, another object state transition diagram can be attached to a link through the Object State Transition attribute to specify another transition network breaks this transition down into further detail.

4.1.1.16 Object State Transition

The Object State Transition class represents the organizing structure of the object state transition diagram. It keeps track of all object states and links that are defined in the diagram. The Object States attribute keeps a list of objects states used in the diagram while the Transition Links attributes keeps a list of the transition links that are defined in the diagram.

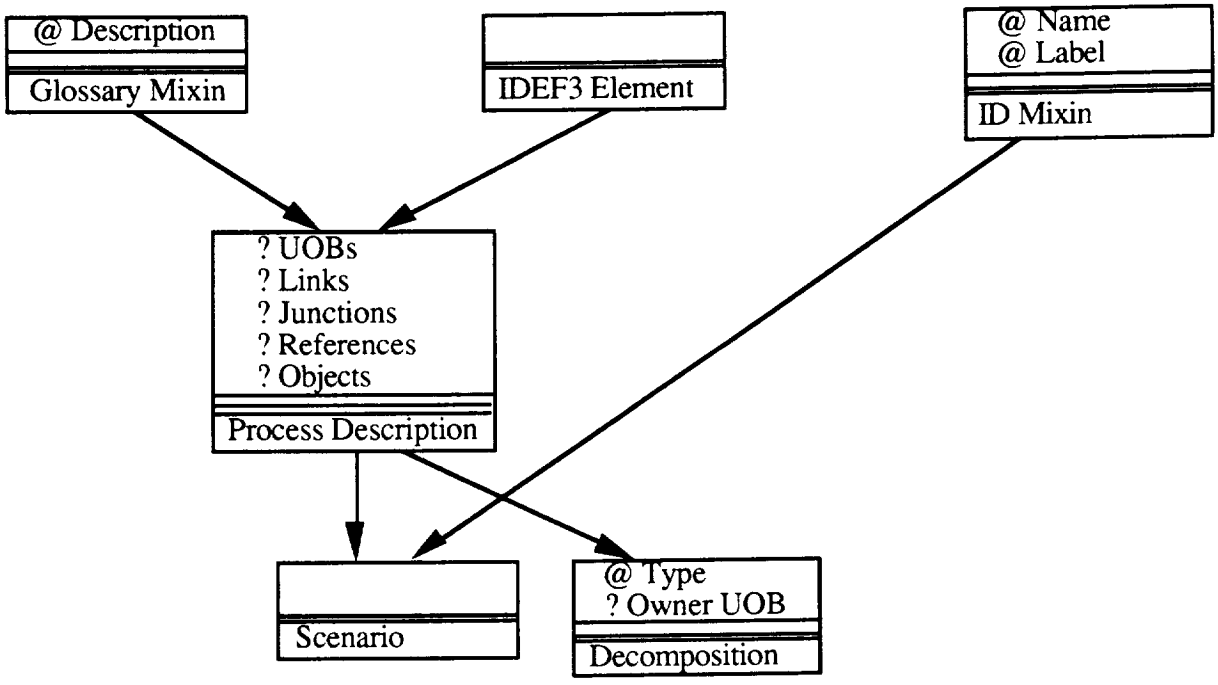


Figure 6. IDEF3 Organization Objects Class Inheritance Diagram

Figure 6 presents the Class Inheritance Diagram for the organizational structures of IDEF3. The most important fact represented here is that both a Scenario and a Decomposition are Process Descriptions. The main difference is that a scenario is a named process description while a decomposition has no name, but is instead referenced by attaching it to a UOB (Owner UOB). A Decomposition also has a Type attribute that a scenario does not require. Also notice that the Scenario class inherits all of its characteristics from the Process Description class and the ID Mixin class.

Figure 7 presents the class inheritance diagram for the process description elements of IDEF3. To begin with, all five objects (Elaboration, Link, Junction, UOB, and Referent) inherit from IDEF3 Element and Glossary Mixin. More interestingly, notice that, of these objects, only UOBs and Referents inherit from ID Mixin. As a result, instances of the other classes must be references through other means. Finally, also note that UOB, Junction, and Referent inherit from the Linkable Object class, indicating that instances of these classes can be specified as the front or back object of a link.

Finally, Figure 8 presents the class inheritance diagram for the classes that define the Object State Transition diagrams. This is a pretty uneventful diagram as all five classes, Object State, Object, Transition Link, Object State Transition, and Attribute Value Pair, inherit from Glossary Mixin, IDEF3 Element, and ID Mixin.

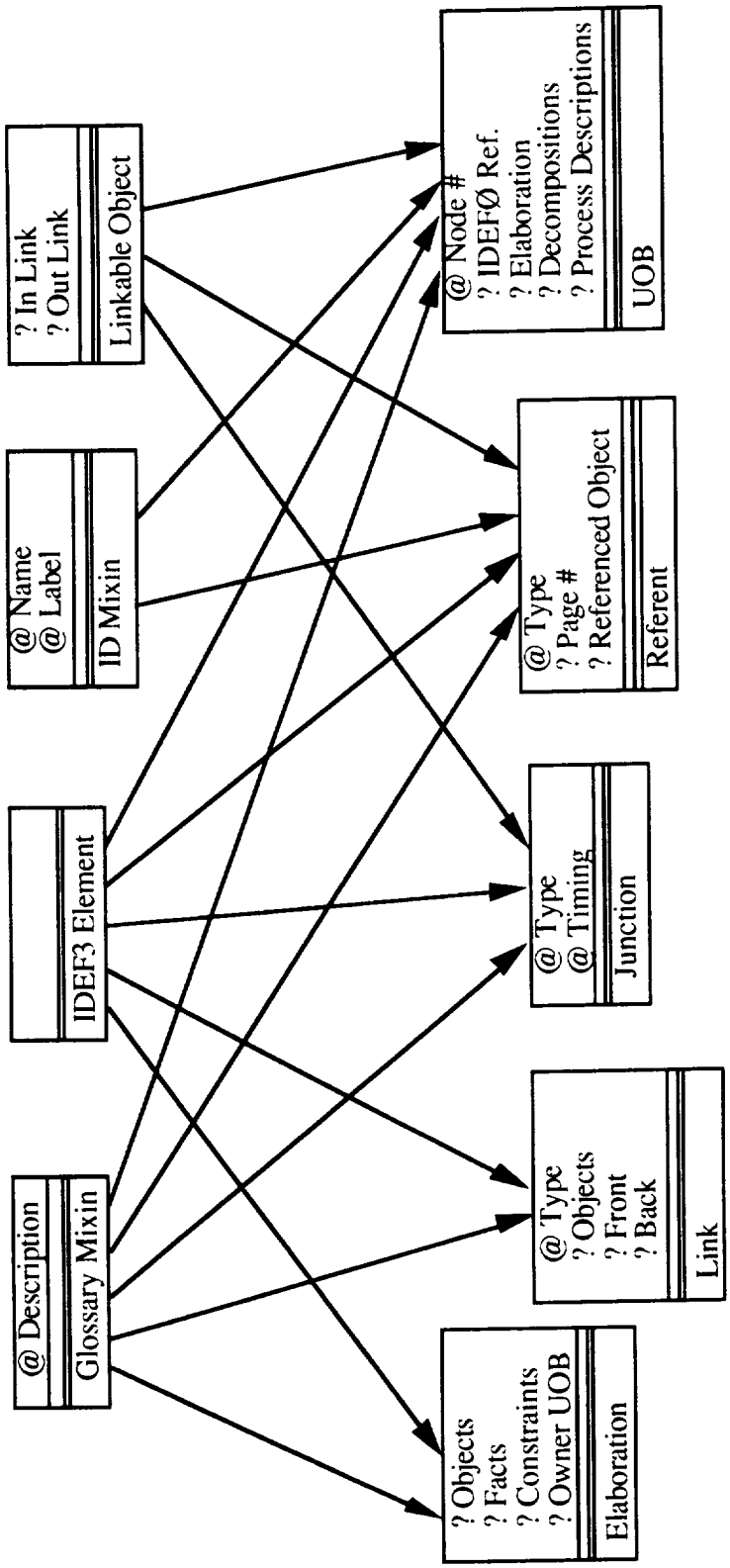


Figure 7. Basic IDEF3 Objects Class Inheritance Diagram

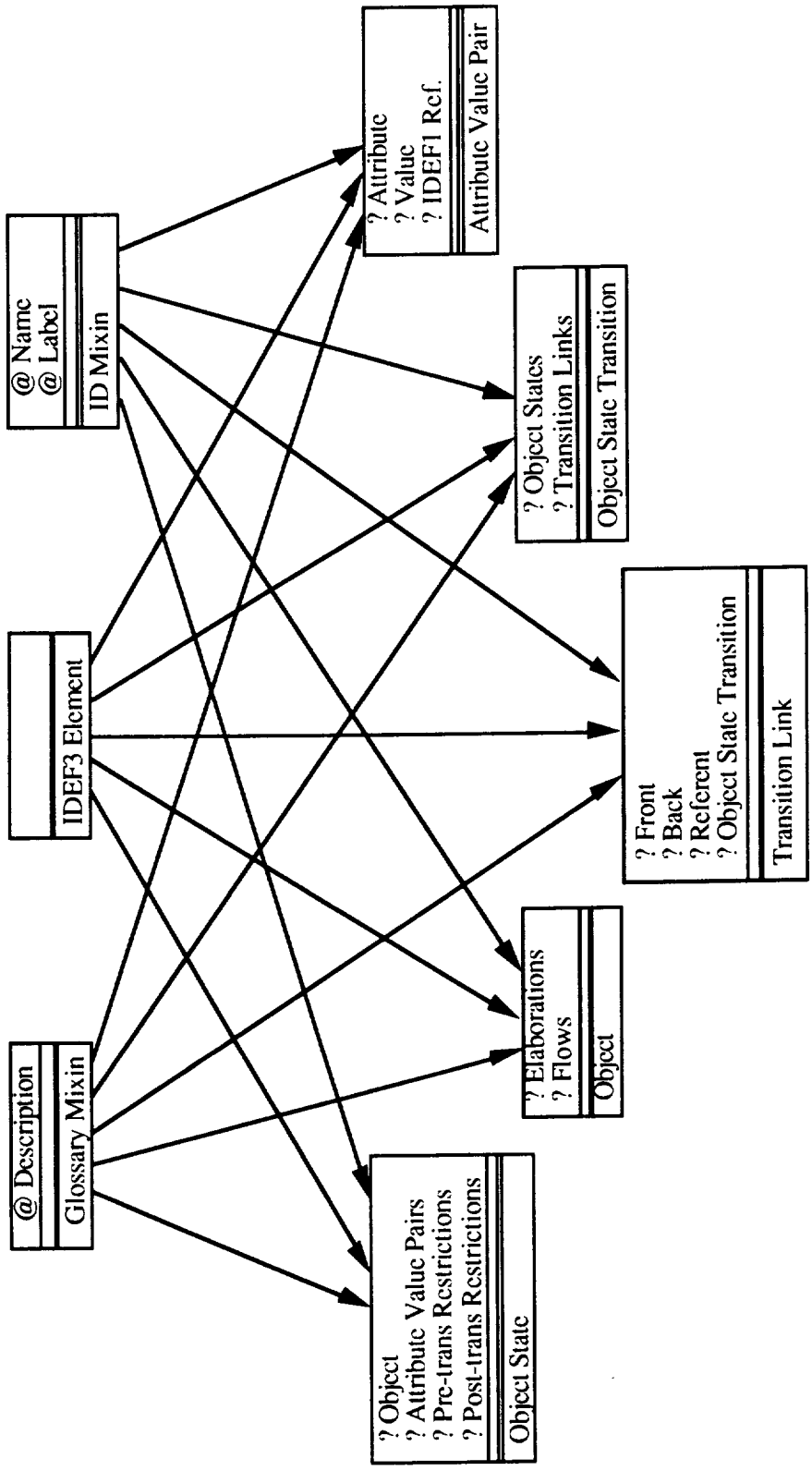


Figure 8. Object State Transition Objects Class Inheritance Diagram

4.1.2 IDEF3 User Interface and Constraint Enforcement

At this point, the focus of the IDEF3 design description shifts from the objects to be manipulated by the system to the processes that are performed to actually manipulate those objects. Appropriately, IDEF3 process flow descriptions will be used to relate these processes. It is hoped that these process descriptions will give the reader a feel for what it will be like to use the IDEF3 tool. To assist in achieving this goal, some UOBs will have Referents attached to them that point to representative screens. These screens will be drawings of the screen that the user will encounter at that point in the development process. The discussion of the IDEF3 descriptions will indicate any constraints that must be satisfied by a particular process.

NOTES:

(1) In these IDEF3 descriptions, the numbering convention of Units of Behavior have been changed. Because every UOB in these descriptions have only one decomposition, if any, the decomposition number after the 'v' in the UOBs has been dropped. For example, if a UOB, 1, had a decomposition with a UOB numbered 3 in the decomposition, that UOBs node number would have been 1.v1.3. Instead, in the following diagrams, the node number would be 1.v.3. Also, due to the diagram size, the prefix of the UOBs is being removed and placed in the top left corner of the diagram. This notation will simplify the numbering scheme of these diagrams as the number of decompositions increases.

(2) The UOBs that have shadowed boxes are UOBs that have decompositions defined for them.

(3) A special character, ¥, is placed in the upper left corner of a UOB to indicate that that particular UOB represents an explicit user action. UOBs without that character represent general procedures of processes performed by the tool.

(4) Throughout the process descriptions, certain UOBs that indicate user actions have referents attached to them labelled "See IDEF3 Screen #". Sketches of the proposed screens for the IDEF3 tool have been created and can be found at the end of this section. These referents are refering to these screens so that

the reader can get a feel for what the system will look like when performing certain operations.

There is one aspect of the IDEF3 tool that was not described in the process descriptions because of the added complexity it would add to the descriptions. During the execution of any of the process supported by the tool, it is possible for the user to abort out of the process. The effect of this abort is to return to the state that existed before the aborted process was started.

To begin with, Figure 9 shows the top level process for the User/IDEF3 Tool Interaction Scenario and its decomposition. Once the tool has been started, a new description database is created or an existing description database is opened. The user then decides what mode of operation (see Section 2.1) they wish to work in and then begins the development of the process flow description and the object state transition diagrams. The UOBs 1, 1.v.1, and 1.v.2 all reference activities defined in the IDEFØ model of IDEF3 that can be found in "IDEF3 and IDEF4 Automation System Requirements Document and System Environment Models".

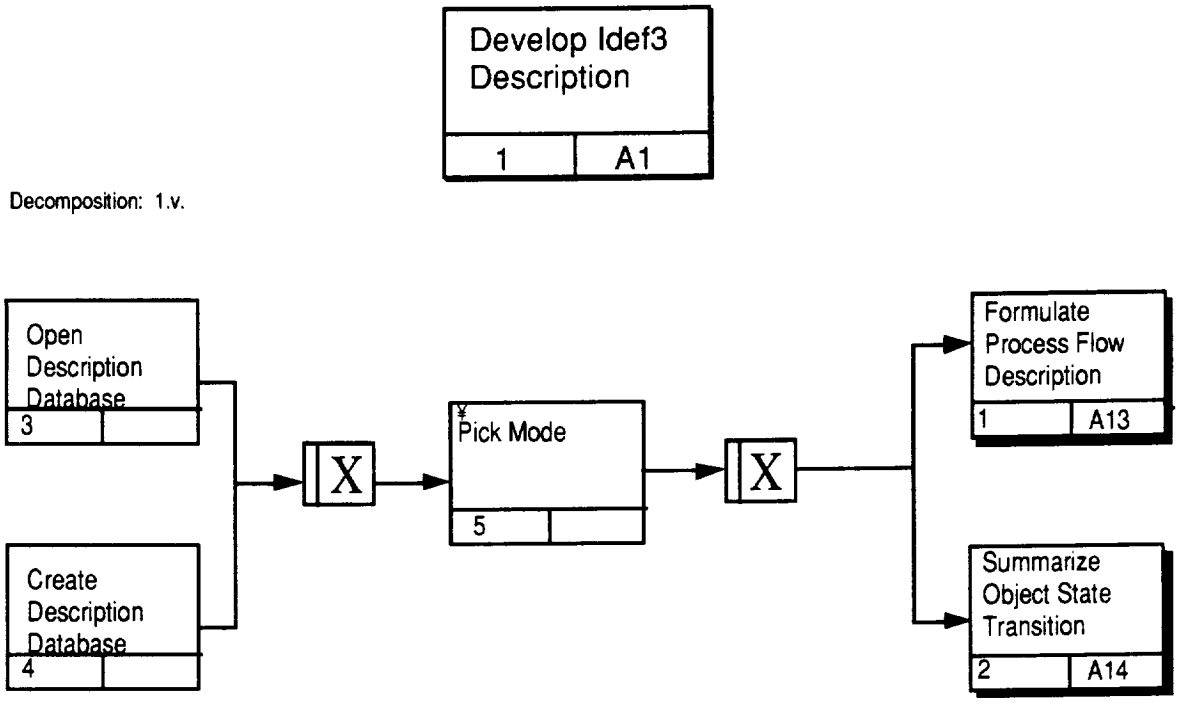


Figure 9. User/IDEF3 Tool Interaction Scenario

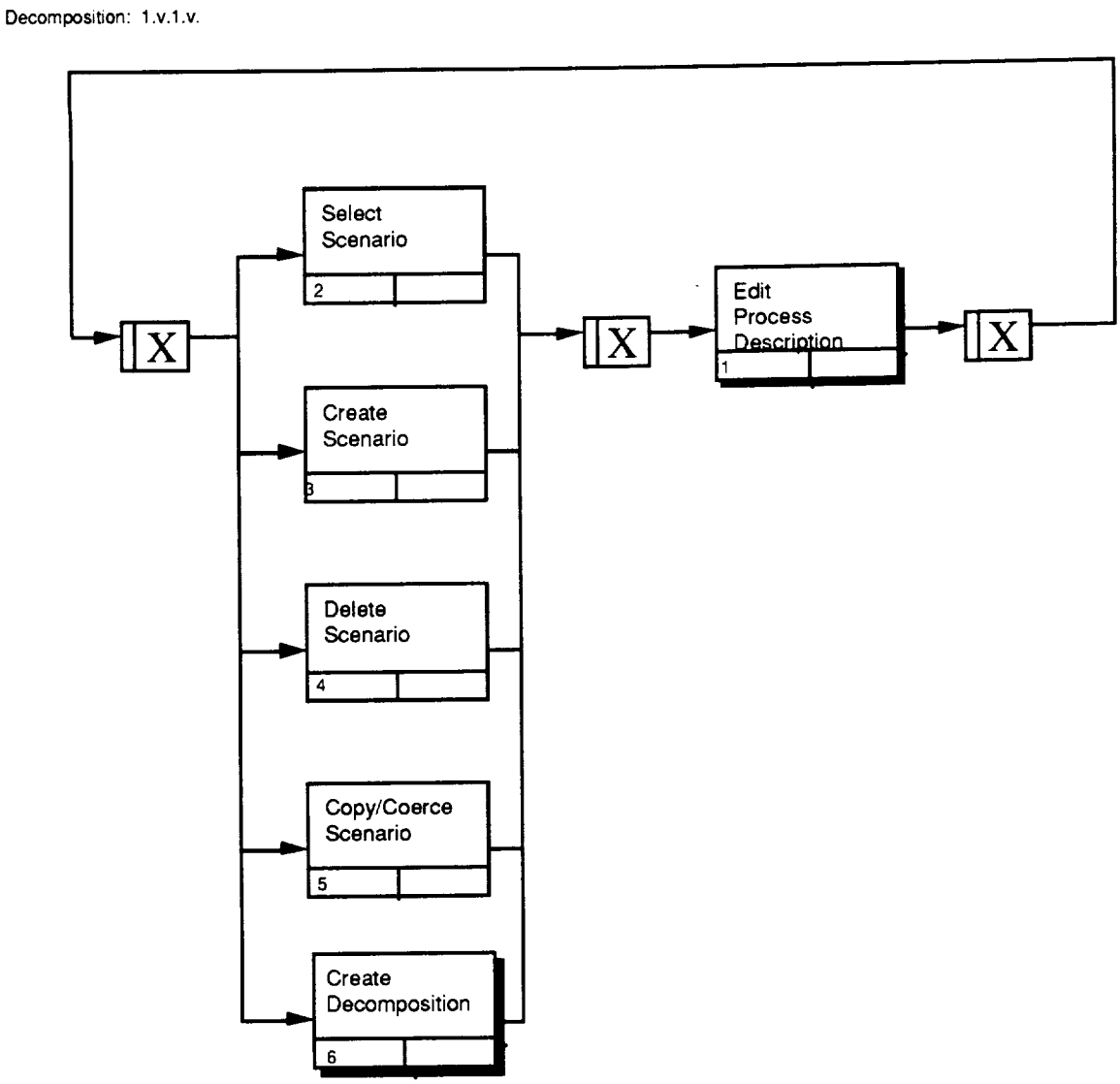


Figure 10. Decomposition of Formulate Process Flow Description

Figure 10 displays the decomposition of the Formulate Process Flow Description process. At this point, the user can create an entirely new scenario, select an existing scenario for editing purposes, delete a sceanario, copy an existing scenario or coerce a scenario into a decomposition, or can create a new decomposition. Once one (because of the XOR junction) of these actions is performed, the user can begin the process of editing a process description.

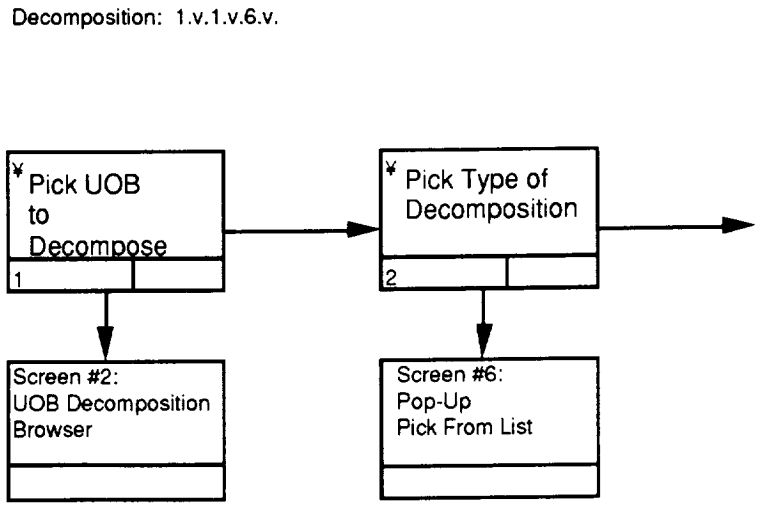


Figure 11. Decomposition of Create Decomposition

Figure 11 displays the decomposition of the Create Decomposition process. In performing this operation, the user must specify the UOB for which the diagram is being created and the type of decomposition being created (Objective View or View). Also notice the references to the IDEF3 screens that will be active during the execution of this process.

Figure 12 presents the decomposition of the Edit Process Description UOB. An important fact to note about this description is that there is an implicit loop around this particular process. It would not be very useful to be able to execute only a single command. Instead, once a command has been executed, the user can begin processing another command by using this same process. This description also indicates the mode of interaction between the user and the system. There are two different ways to execute a command, by either clicking on a command in the command menu or by typing the command at the command line. Also note the the File (database) and Mode operations are available only through the command menu.

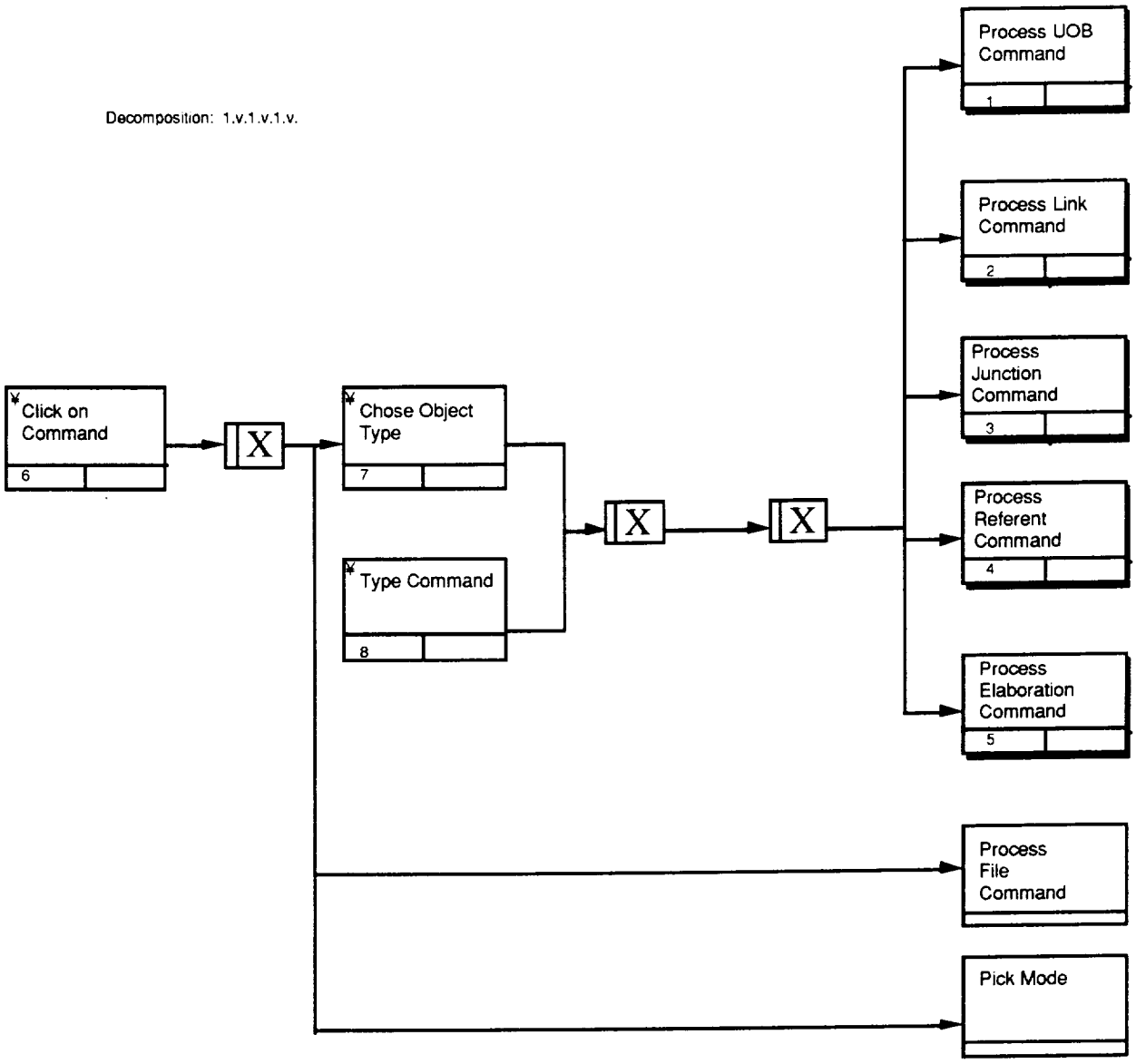


Figure 12. Decomposition of Edit Process Description

Figure 13 displays the decomposition of the Process UOB Command process. A variety of different commands for the manipulation of UOBs will be available, each requiring their own execution process. Notice that the screen reference indicates that these operations will be performed in the Process Description Diagram. Some of these operations will also be available in the UOB Decomposition Browser.

Constraints:
The UOB operations have certain constraints that must be satisfy for the operation to be carried out successfully:

- The Copy UOB operation requires that the name of the UOB be changed.
- When a UOB is deleted from a process description, any Referents that refer to that UOB must be updated to reflect the deletion.

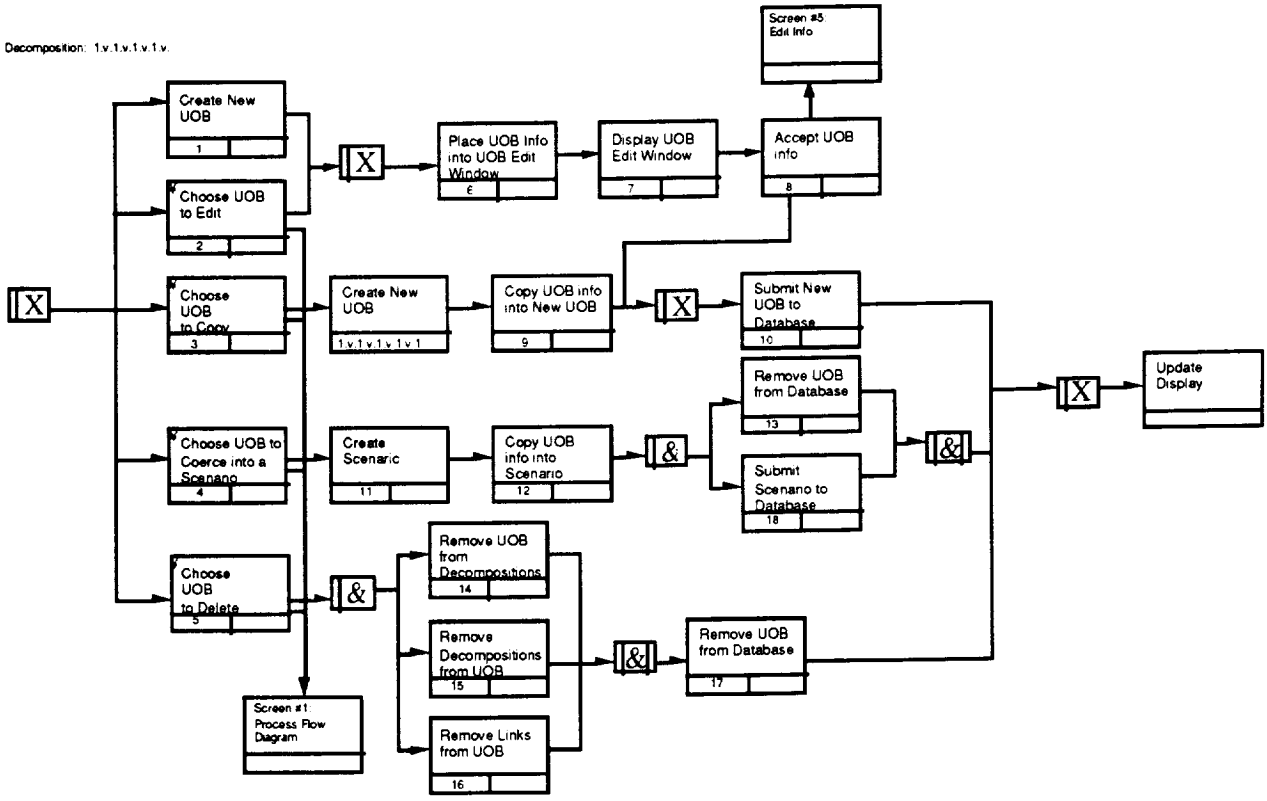


Figure 13. Decomposition of Process UOB Command

Figure 14 displays the decomposition of the Process Link Command. The most significant thing to point out here is that a link will be created between existing objects. As a result, the user selects the objects that are to be related by a link.

Constraint:

- A linkable object can have only one link into itself and one link out of itself. As a result, links can only be created between linkable objects that have not exceeded these limits.
- When a link is created between a Junction and some other objects, the resulting structure must be analyzed to determine if the structure is valid. For example, if an fan-out XOR junction is eventually linked with a Synchronous AND junction, the structure is invalid since the AND requires multiple processes to

terminate while the XOR allows only one process to begin execution. This situation cannot be allowed.

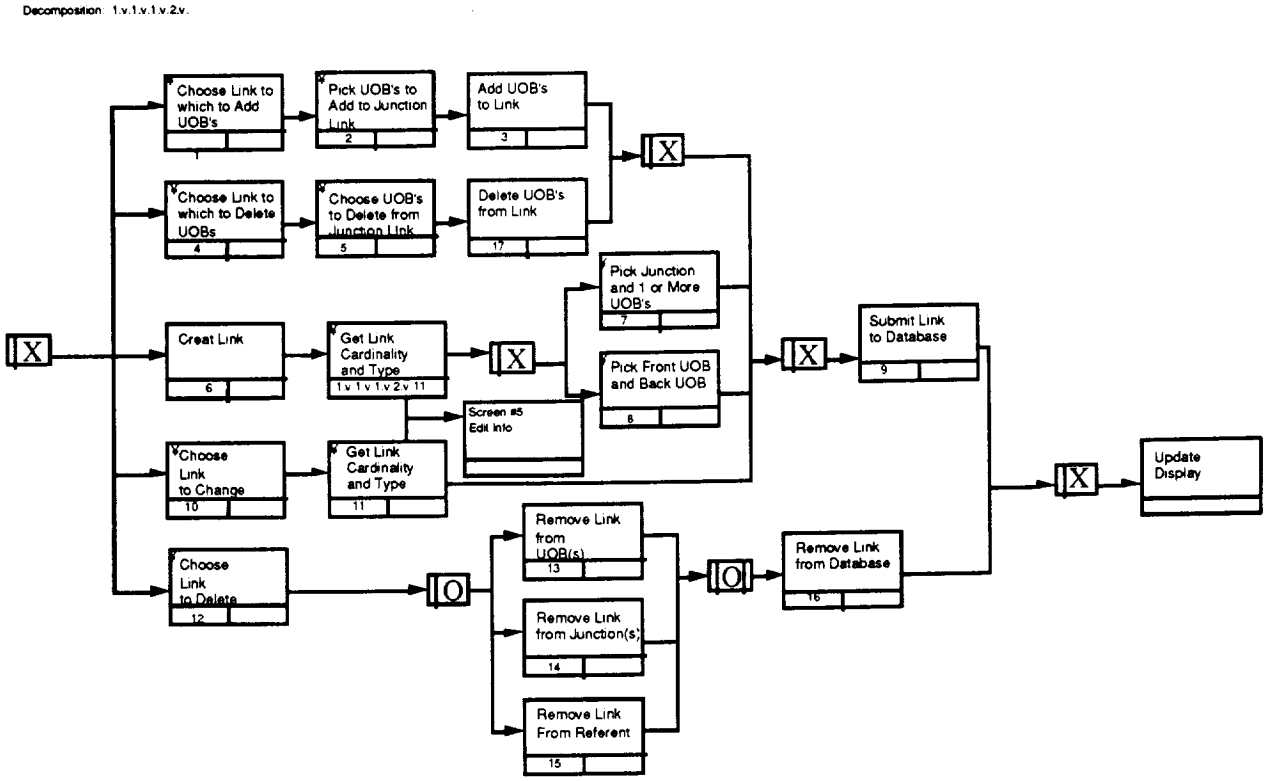


Figure 14. Decomposition of Process Link Command

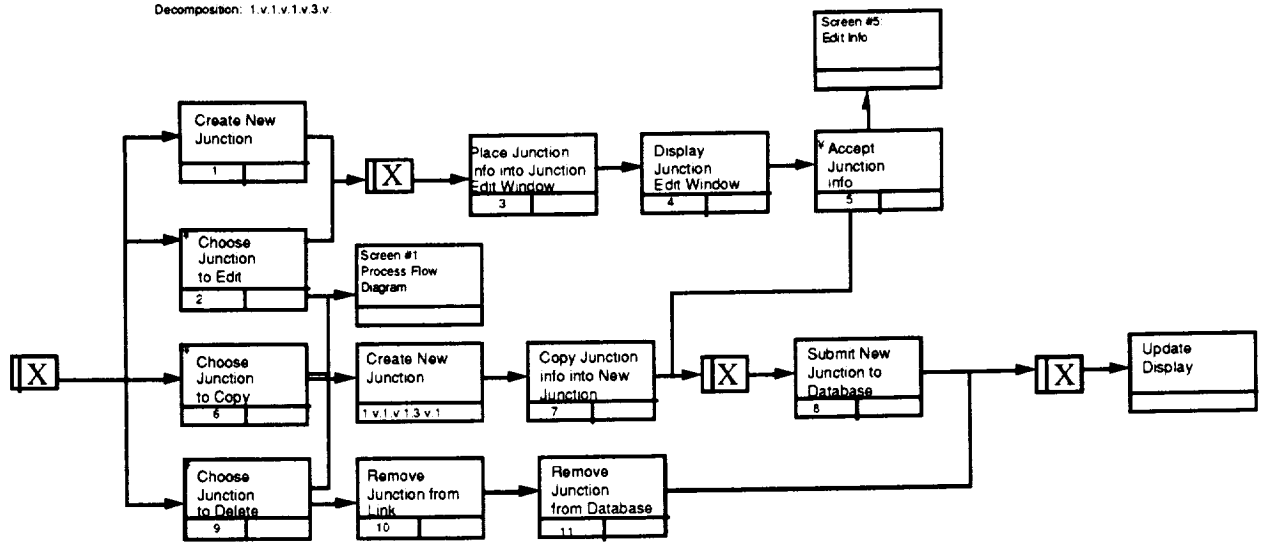


Figure 15. Decomposition of Process Junction Command

Figure 15 displays the decomposition of the the Process Junction Command. The junction commands are relatively straightforward as is obvious by the relatively simple process diagram.

Constraints:

- When a Junction is deleted, those links that are connected to the junction should also be deleted.

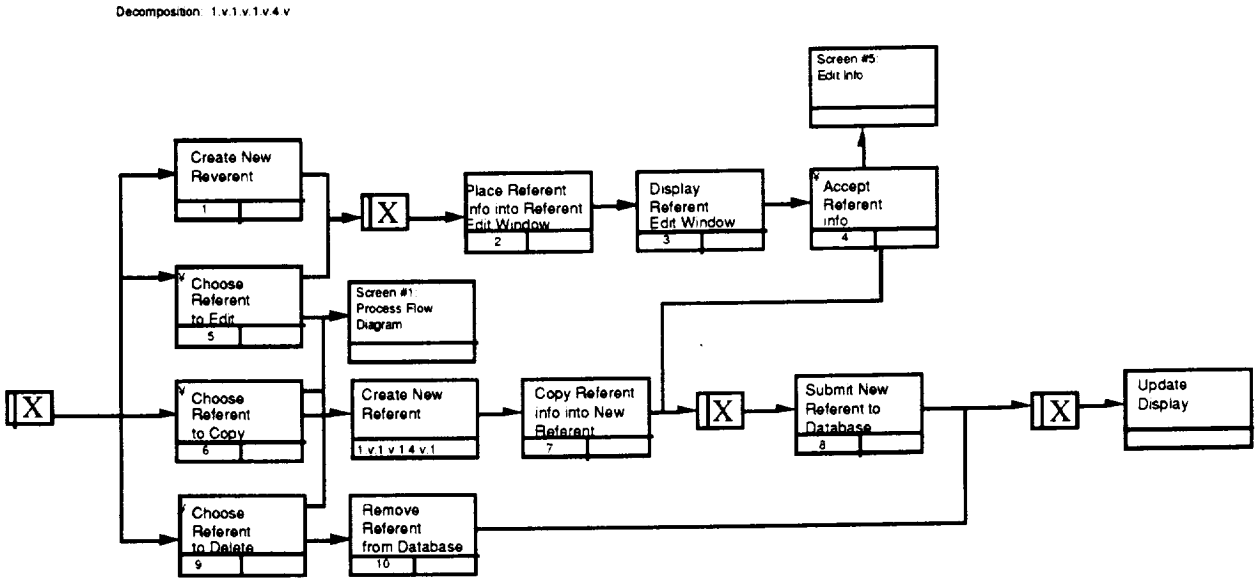


Figure 16. Decomposition of Process Referent Command

Figure 16 displays the decomposition of the Process Referent Command. These operations will be available in any of the IDEF3 modes since Referents can be used in many different contexts.

Figure 17 displays the decomposition of the Process Elaboration Command. Though there is no Referent to indicate it in this description, these operations can be performed in the Process Description mode or in the UOB Decomposition Browser.

Constraints:

- Only one Elaboration can be associated with any given UOB.
- When an Elaboration is deleted from a UOB, if that UOB has an Objective View Decomposition, that decomposition must be redefined as a simple View Decomposition.

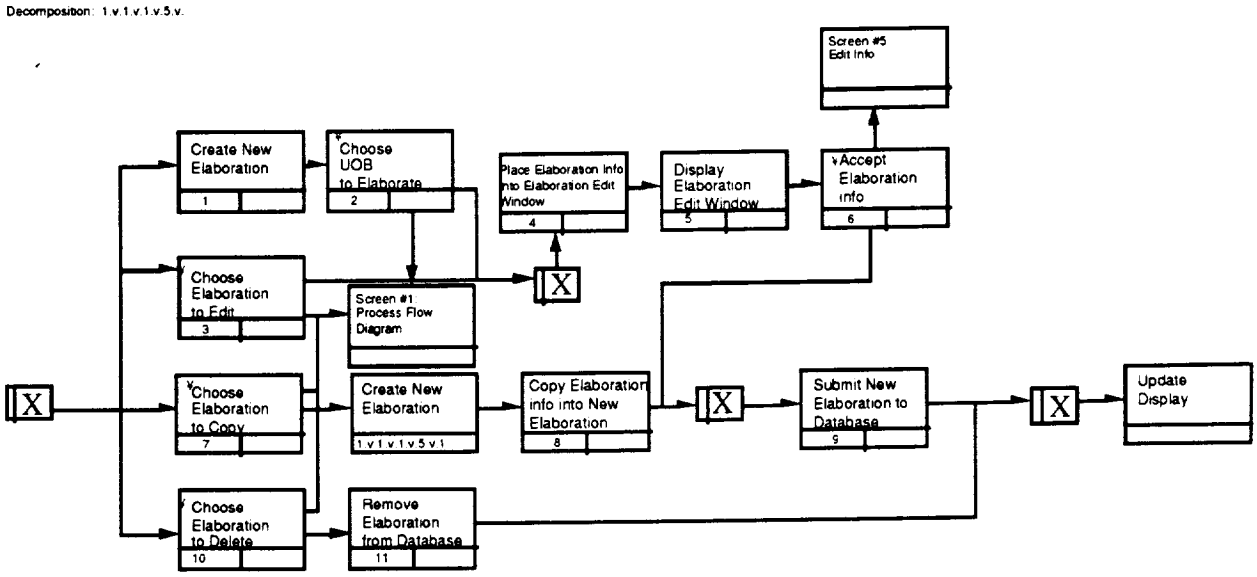


Figure 17. Decomposition of Process Elaboration Command

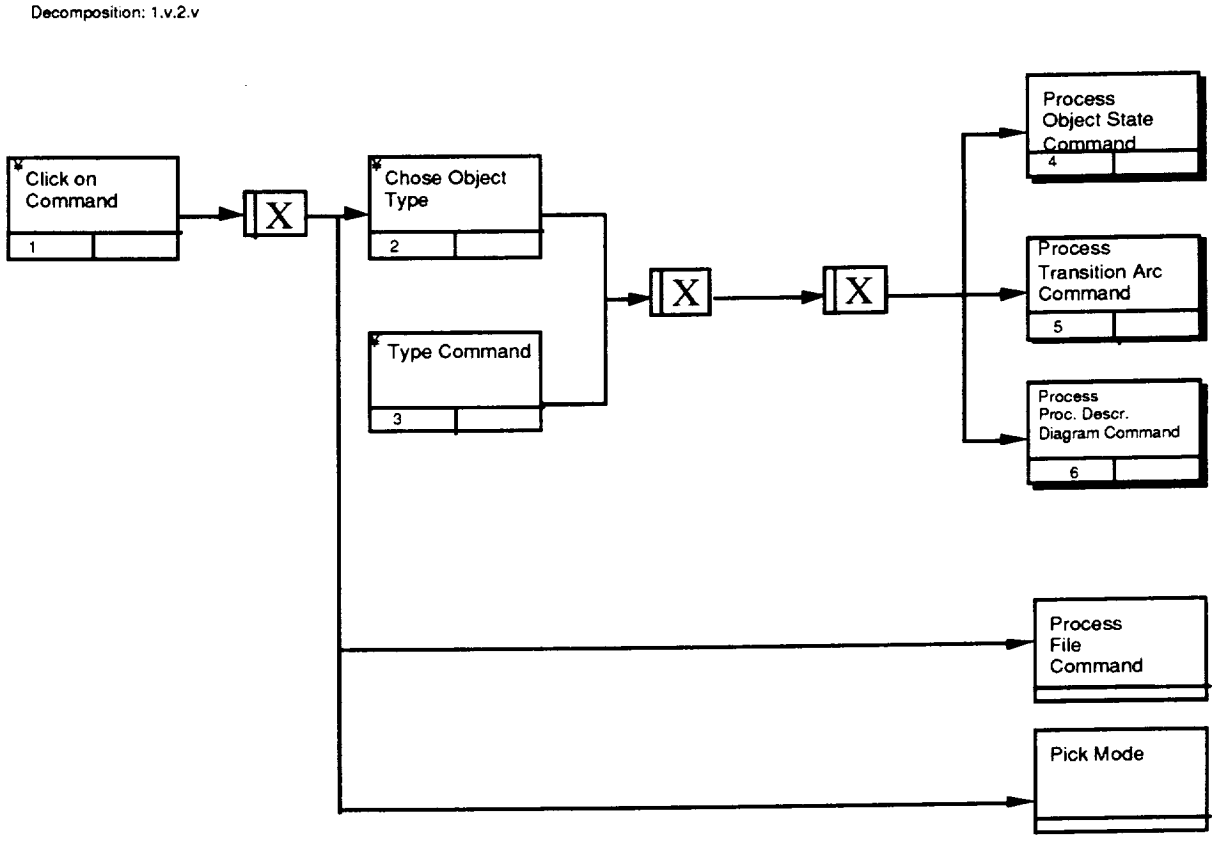


Figure 18. Decomposition of Summarize Object State Transition

Figure 18 presents the decomposition of the Summarize Object State Transition process. The operations at this level are very similar to the process flow operations. The only difference is the different modes of operations that are used for Object State Transition Diagrams.

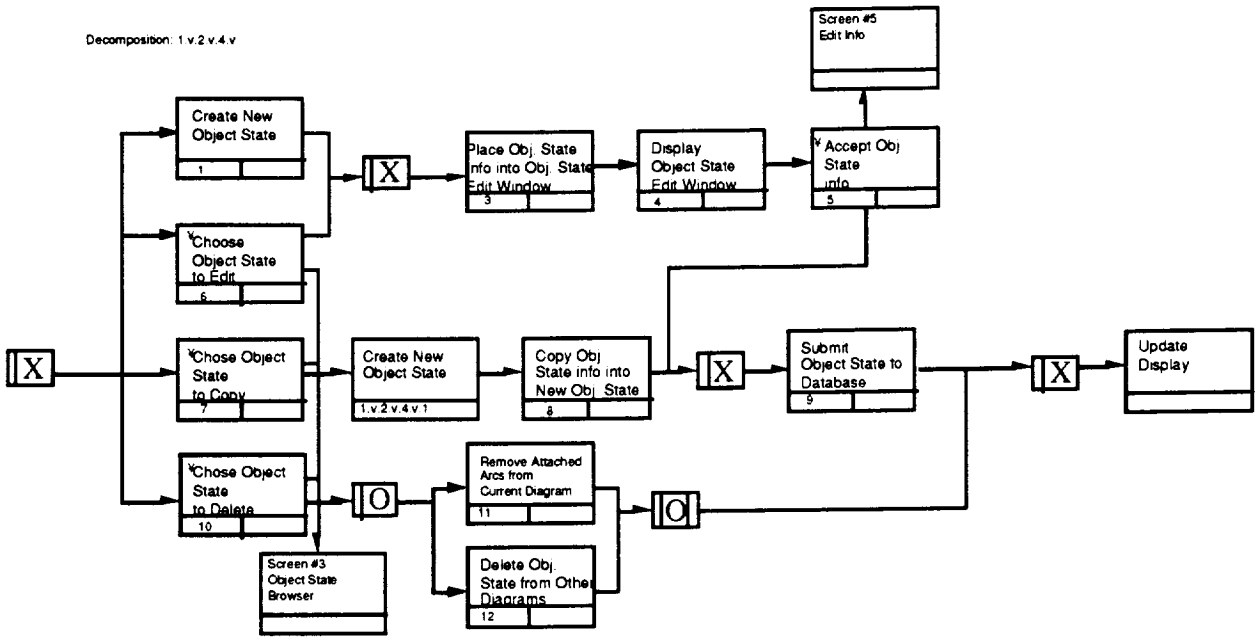


Figure 19. Decomposition of Process Object State Command

Figure 19 displays the decomposition of the Process Object State Command process. The operations on an Object State in a Transition Diagram are very similar to the operations that would be performed to a UOB in a process description diagram. These operations will be available in the Object State Transition Diagram mode or in the Object State Browser.

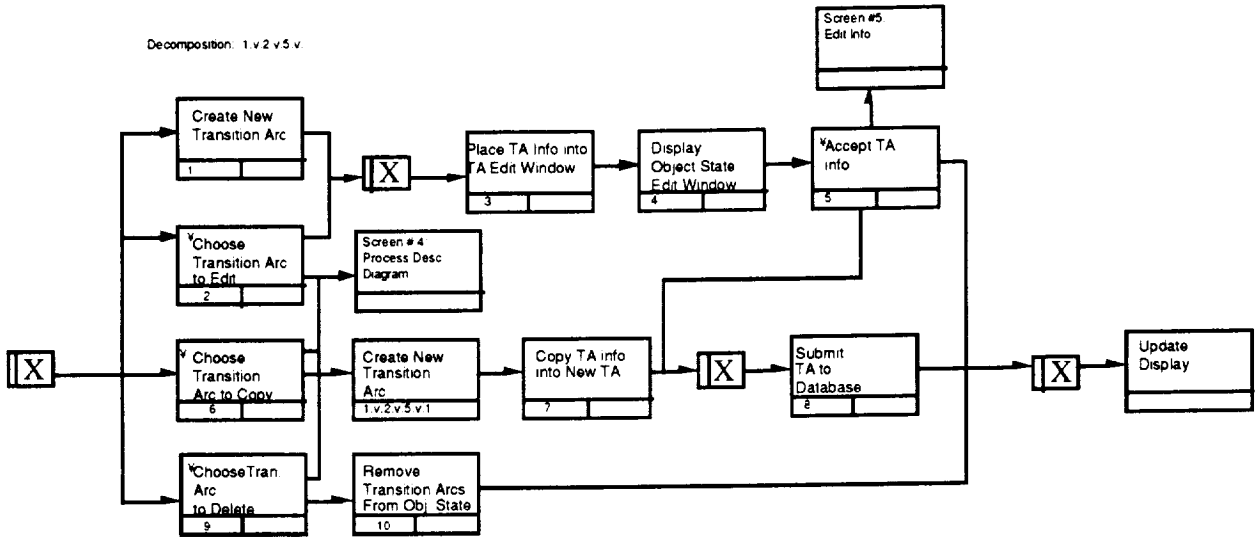


Figure 20. Decomposition of Process Transition Arc Command

Figure 20 shows the decomposition of the Process Transition Arc Command. Its operations are very similar to the Link operations in that the arc must be specified between existing object states. These operations will be available only in the Object State Transition Diagram mode.

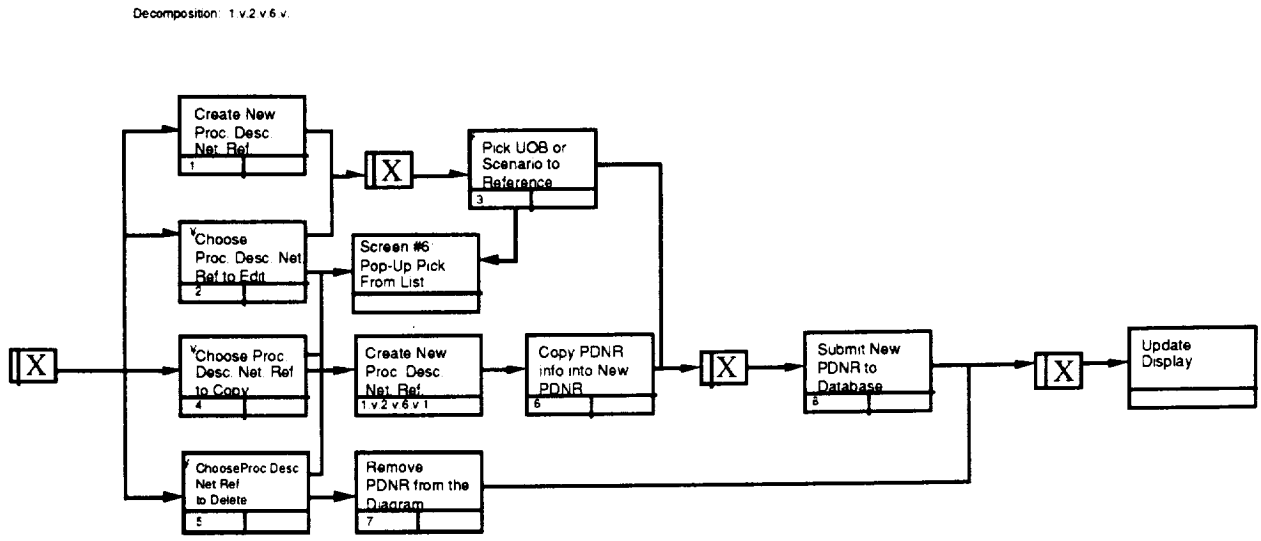


Figure 21. Decomposition of Process Process Description Diagram Command

Figure 21 displays the decomposition of the Process a Process Description Diagram Command. The purposes of these operations is

to attach a process description to a transition arc. The semantic effect of doing this is to indicate that the process attached to the link will effect the transition between object states linked by the arc.

Figures 22 through 27 display potential screens that will exist within the IDEF3 system and are the screens that are referenced within the process descriptions. These screens are meant only to give a rough idea of how the tool screens will appear. Figure 22 shows a potential Process Flow Diagram screen. Figure 23 displays the Unit of Behavior Decomposition Browser. Figure 24 shows the Object State Browser. Figure 25 shows the Object State Transition Diagram screen. Figure 26 displays an Input/Edit Pop-up Window. Finally, Figure 27 displays a Pick From List Pop-up Window.

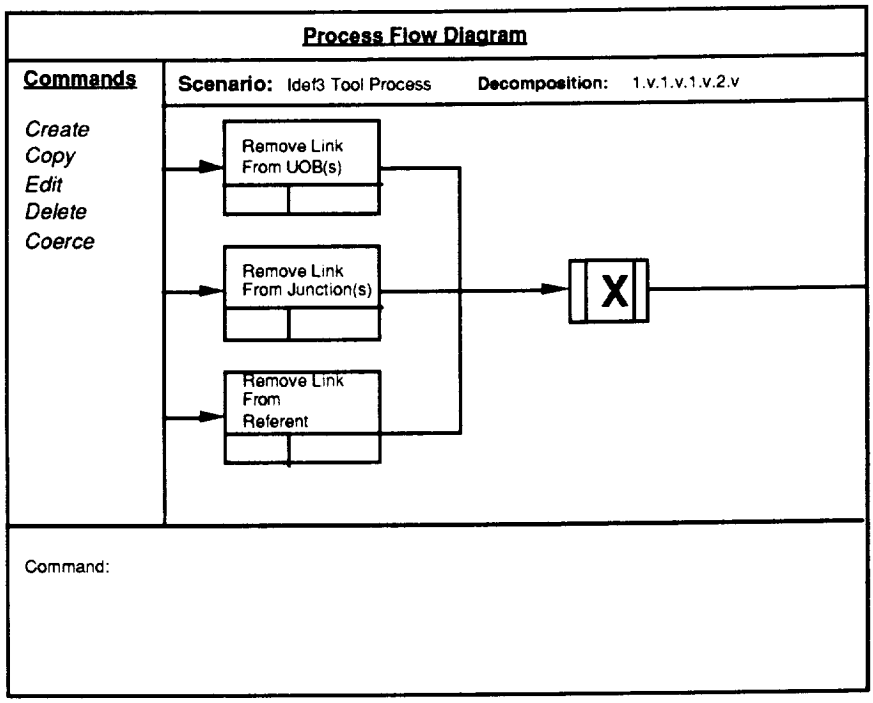


Figure 22. IDEF3 Screen #1 - Process Flow Diagram

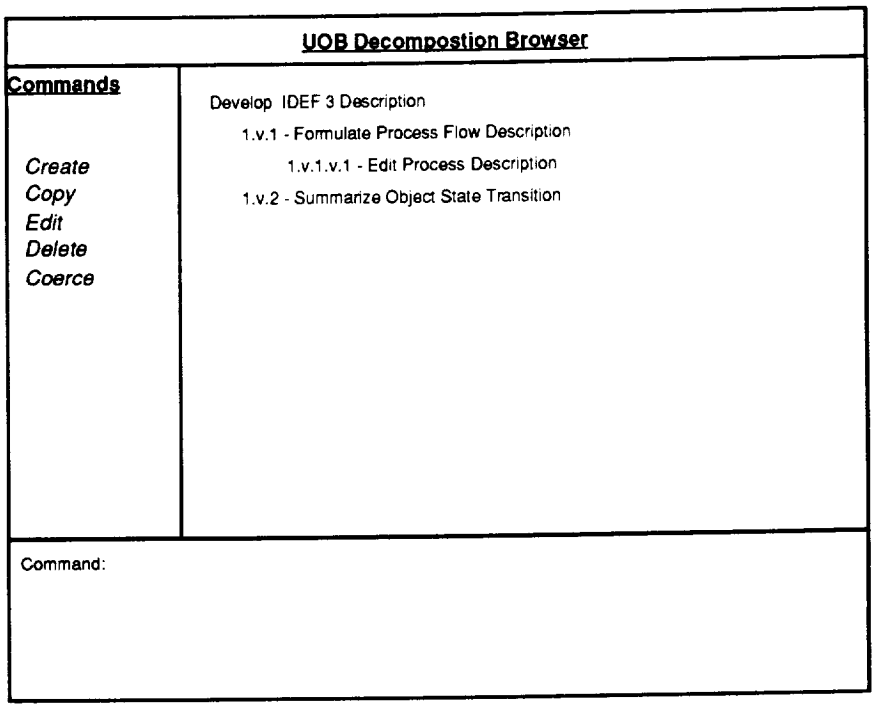


Figure 23. IDEF3 Screen #2 - UOB Decomposition Browser

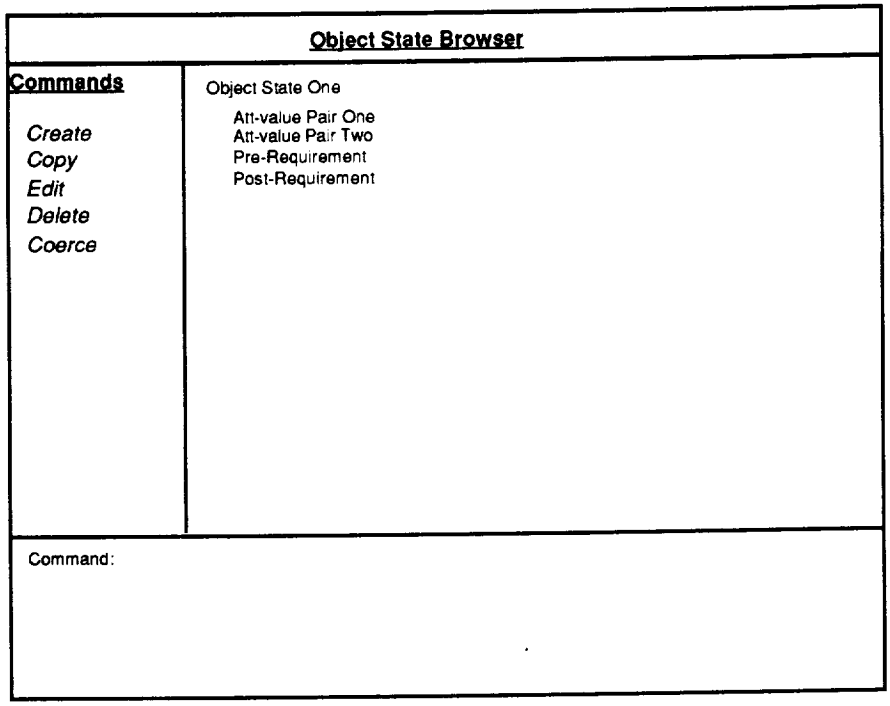


Figure 24. IDEF3 Screen #3 - Object State Browser

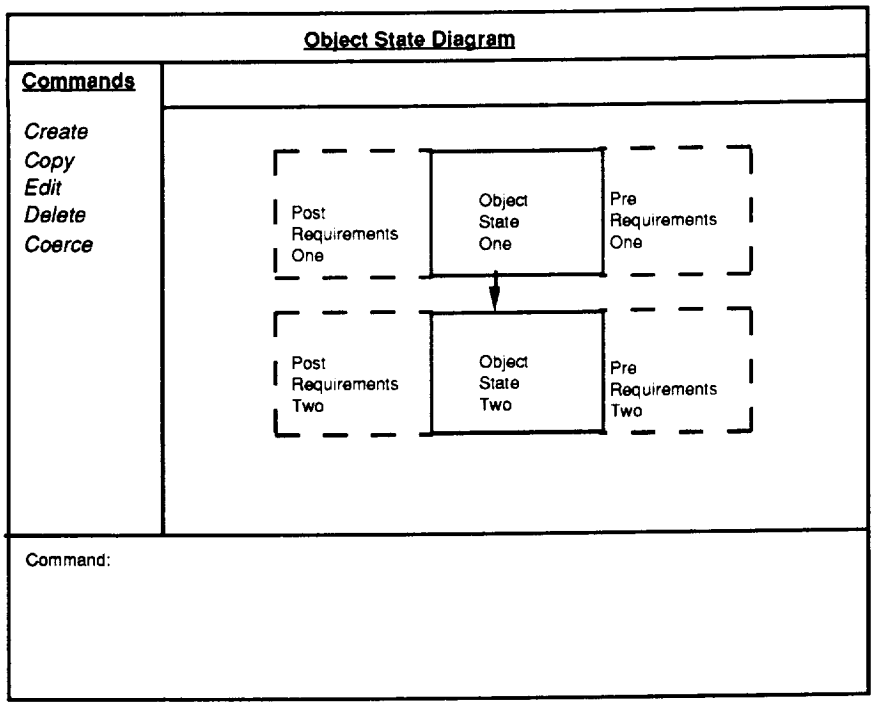


Figure 25. IDEF3 Screen #4 - Object State Transition Diagram

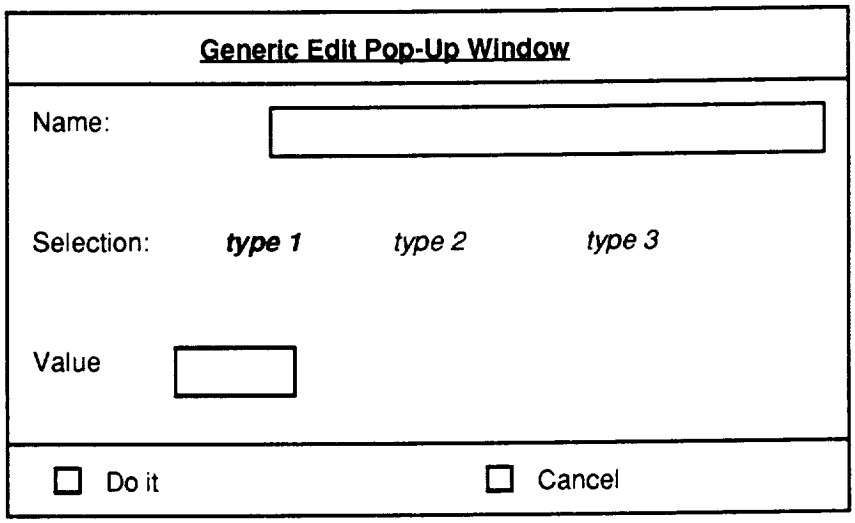


Figure 26a. IDEF3 Screen #5a - Edit Pop-up Window

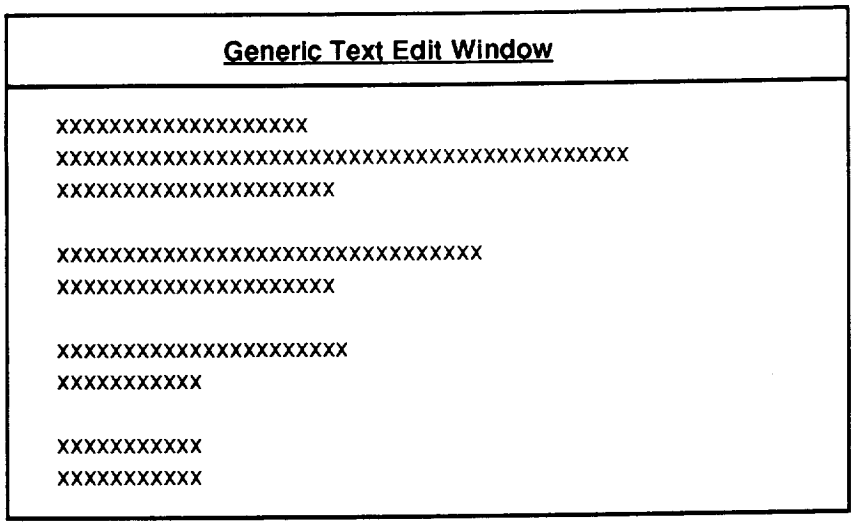


Figure 26b. IDEF3 Screen #5b - Text Edit Pop-up Window

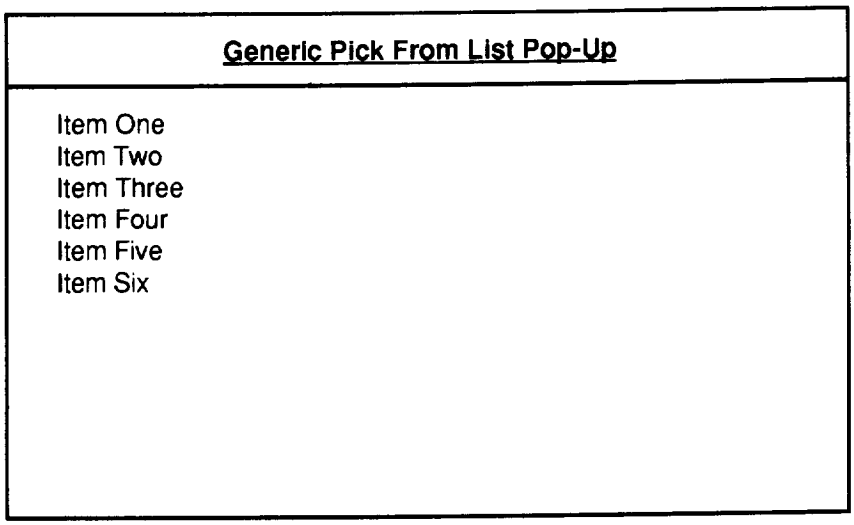


Figure 27. IDEF3 Screen #6 - Pick From List Pop-up Window

4.2 IDEF4 Design Components

At this time, a more detailed analysis of the IDEF4 design will be presented. This discussion begins with the definition of the classes that will exist within the IDEF3 system along with their inheritance relationships to each other. After the classes have been defined, the IDEF4 processes that manipulate instances of those classes will be described. These processes will give a feeling for how the user will

interact with the IDEF4 tool to effectively design an object oriented system.

4.2.1 IDEF4 Data Structures

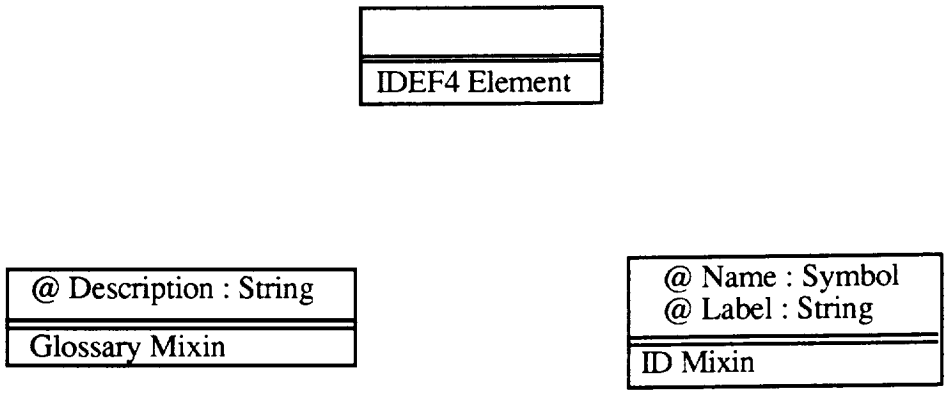


Figure 28. IDEF4 Mixin Objects Type Diagram

Figure 28 displays the type diagram of three classes that will be used for 'mixin' purposes. The term 'mixin' serves the same purpose here as it did in the discussion of the IDEF3 mixin classes. These classes represent a group of features and functionality that are common to several classes in the system. As a result, these common elements are pulled out into their own classes so that the code to implement the operations will be developed only once.

4.2.1.1 IDEF4 Element

This class represents the basic functionality required for the manipulation, display, and presentation of any element in the IDEF4 system. As yet, no features have been identified, but various display operations have been defined for the IDEF4 elements.

4.2.1.2 Glossary Mixin

This class represents a textual description that will be attached to instances of classes that inherit from this class. The only feature defined for Glossary Mixin is Description, a slot accepting a value of type String. This description captures information explaining the purpose for the object to which the description is attached.

4.2.1.3 *ID Mixin*

This mixin class will be used for identification purposes. Its two attributes, Name and Label will be used to uniquely identify an object. The Name slot takes Symbols as its value while the Label slot accepts Strings as its value. In cases where only the name is required to uniquely identify an object, the label attribute will take a nil value.

Figure 29 displays the type diagram for the remaining classes that will make up the IDEF4 system. These classes map very closely to the IDEF4 entities that were described in the requirements document. One special note should be made on some of the Feature types. Notice that the Front and Back attributes of the Link class, among others, accept values of type t. This notation is used to indicate that the actual type accepted by the attribute is more complex than could be indicated in the type diagram. The actual type of each of these attributes will be discussed with the explanation of the class definition below.

4.2.1.4 *Class Grouping*

The Class Grouping class will be used as a 'mixin' class. Its one attribute, Classes, will maintain a user specified list of classes. The classes main use will be to maintain the list of classes that appear in specific diagrams.

4.2.1.5 *Class*

The Class class defines the basic unit of an IDEF4 design. Each Class has a list of Features Uses and a Class Invariance Data Sheet associated with it as well as a list of parent classes, Superclasses, and a list of child classes, Subclasses. It is through these last two features that the inheritance relationships are maintained in the class submodel.

4.2.1.6 *Method Set*

The Method Set class maps directly to the IDEF3 Method Set. The Contracts attribute maintains the method set's contract specifications while the Mappings attribute links the method set with Class-Feature pairs that are dispatched to this method set.

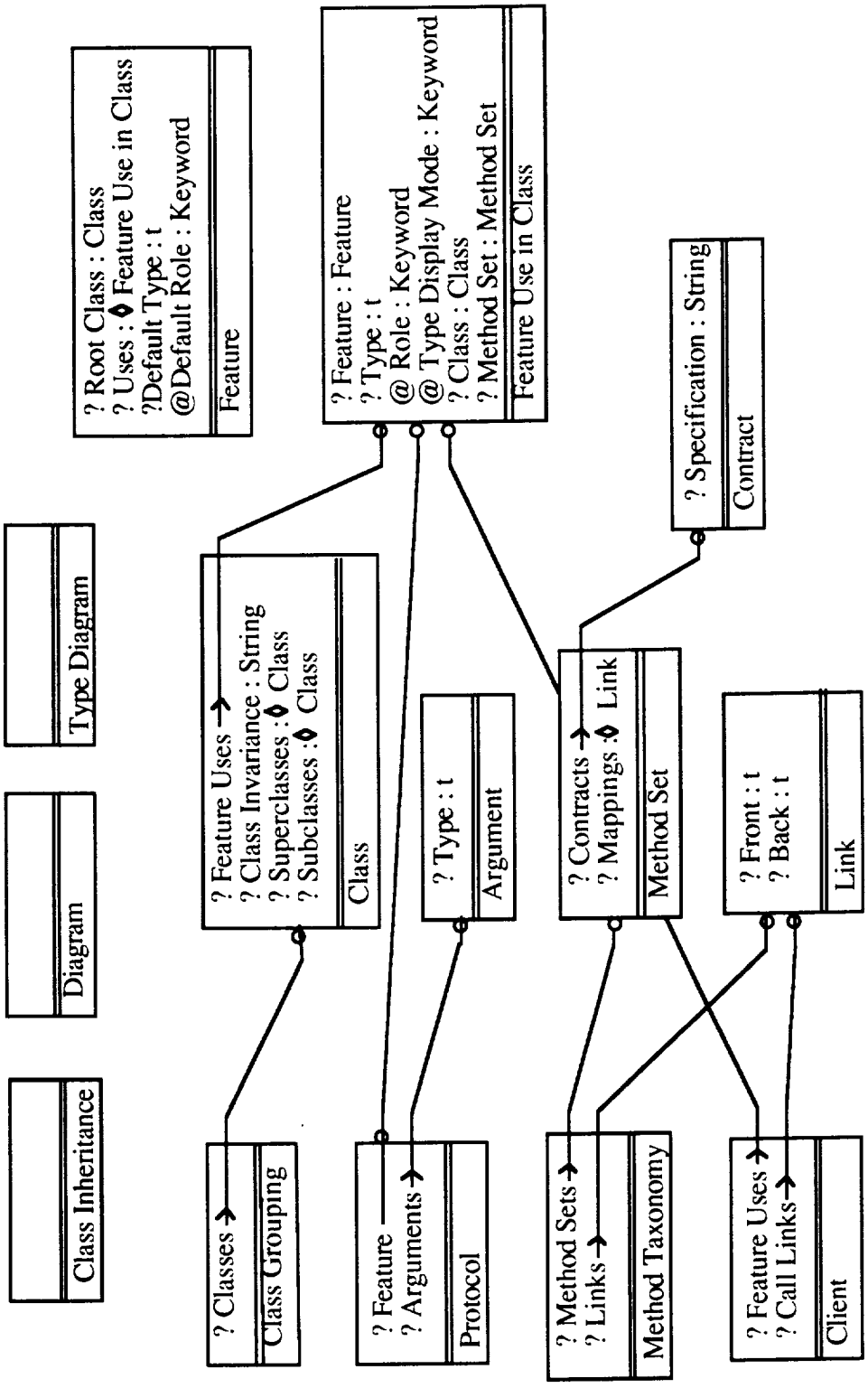


Figure 29. IDEF4 Objects Type Diagram

4.2.1.7 Link

The Link class is used to specify a relationship between two IDEF4 objects. The most notable links will be the subset/superset relationship in Method Taxonomy diagrams and the caller/callee relationship in Client diagrams. The Front and Back attributes indicate the objects that are related by the link. Notice that these attributes accept values of type t, indicating that the attribute type can vary with the context of the link use.

4.2.1.8 Feature

The Feature class represents an IDEF4 class feature. In the tool, though, a feature exists as a separate object, as opposed to a part of a Class, since a feature can be used and redefined in several different classes through inheritance. By creating a feature as a separate object, the functionality of the feature can be represented only once while the tracking of feature use and feature redefinition can be accomplished by other means. The Root Class attribute indicates the class that is said to initially define the feature. It is from this root class that it is determined if later uses of a feature result in redefinitions of the feature. The Uses attributes maintains a list of all classes that use the feature through class inheritance. The Default Type attribute specifies a default feature type to associate with the feature when no other type has been specified. The Default Role attribute serves the same purpose as the Default Type, except that it specifies the role that the attribute will assume (i.e., attribute, function, slot, etc...).

4.2.1.9 Feature Use In Class

The Feature Use In Class class is the method by which the IDEF4 tool will track the inheritance and redefinition of features. This class defines a 'use' link between a class and feature. The Feature attribute references the feature to be used while the Class attribute references the class that will use the feature. When a feature is used, its type and role can be redefined. The Type and Role attributes exist to capture the type and role that the feature is to have for the particular class. A Feature Use also has a Type Display Mode to indicate how the feature type should appear in a Type Diagram and Method Set attribute that maps this Class-Feature pair to a specific method set.

4.2.1.10 Argument

The Argument class is used to represent information that will appear in a Protocol Diagram. IDEF4 allows a protocol to be defined for features of classes. This protocol specifies a list of arguments to the feature as well as the types of those arguments. The argument class captures the name of an argument to a protocol along with the type of that argument.

4.2.1.11 Contract

The Contract class provides the definition of a Method Set. When a feature maps to a method set, it can attach a requirement that must be fulfilled by the method set. This requirement is called a contract. A contracts only attribute is its Specification, a string of information that describes the contractual requirement for the feature.

4.2.1.12 Diagram

The Diagram class represents a graphical display object within the IDEF4 system. It has no features of its own but will provide basic functionality required for the display of the more specific IDEF4 diagrams.

4.2.1.13 Class Inheritance

The Class Inheritance class represents the IDEF4 Class Inheritance Diagram. This class has no owned features, but does inherit features as will be discussed later in this section.

4.2.1.14 Type Diagram

The Type Diagram class represents the IDEF4 Type Diagram. This class has no owned features, but does inherit features as will be discussed later in this section.

4.2.1.15 Protocol

The Protocol class specifies the protocol diagram for a particular feature use. The Feature attribute accepts a Feature Use in Class object as its value that will define the feature/class pair for which the protocol is being defined. The Arguments attribute is simply a list of Argument objects that define the argument list of the protocol.

4.2.1.16 Method Taxonomy

The Method Taxonomy class organizes the method taxonomy diagrams. This class maintains the list of method sets that will appear in the diagram (the Method Set attribute) and the list of links that specify the taxonomic relationships between the method sets (the Links attribute).

4.2.1.17 Client

The Client class organizes the IDEF4 Client diagrams. This class tracks the list of feature/class pairs that are to appear in the diagram (the Feature Uses attribute) as well as a list of the caller/callee links that exists between the class/feature pairs (the Call Links attribute).

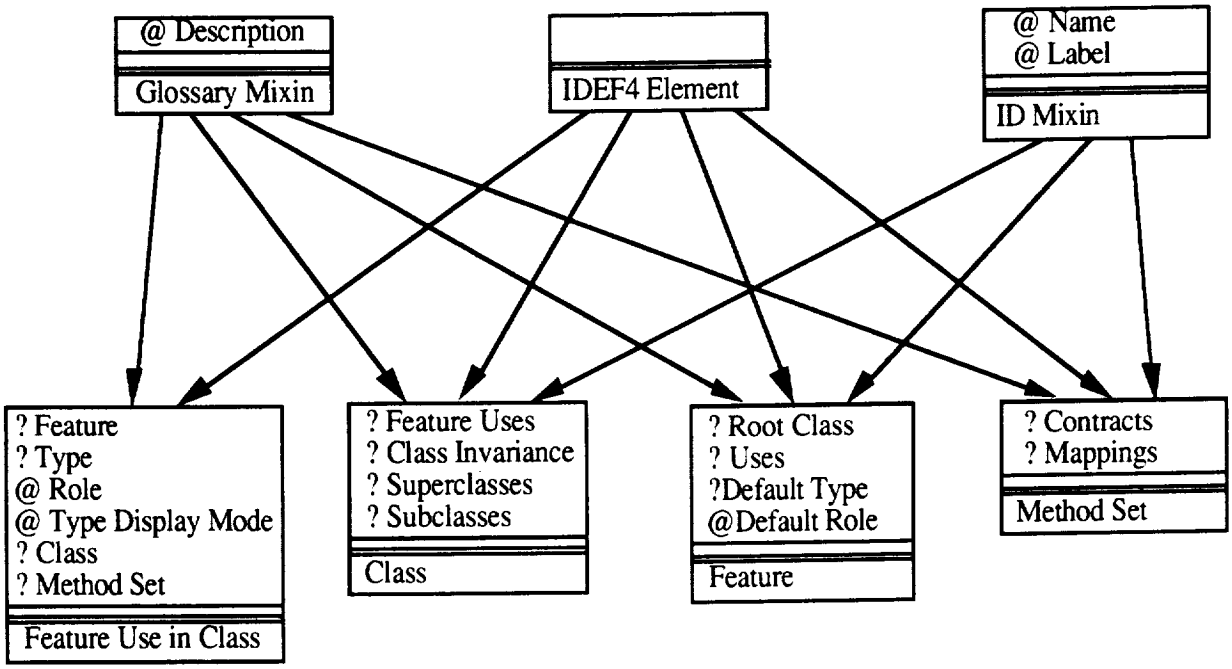


Figure 30. IDEF4 Basic Object Class Inheritance Diagram 1

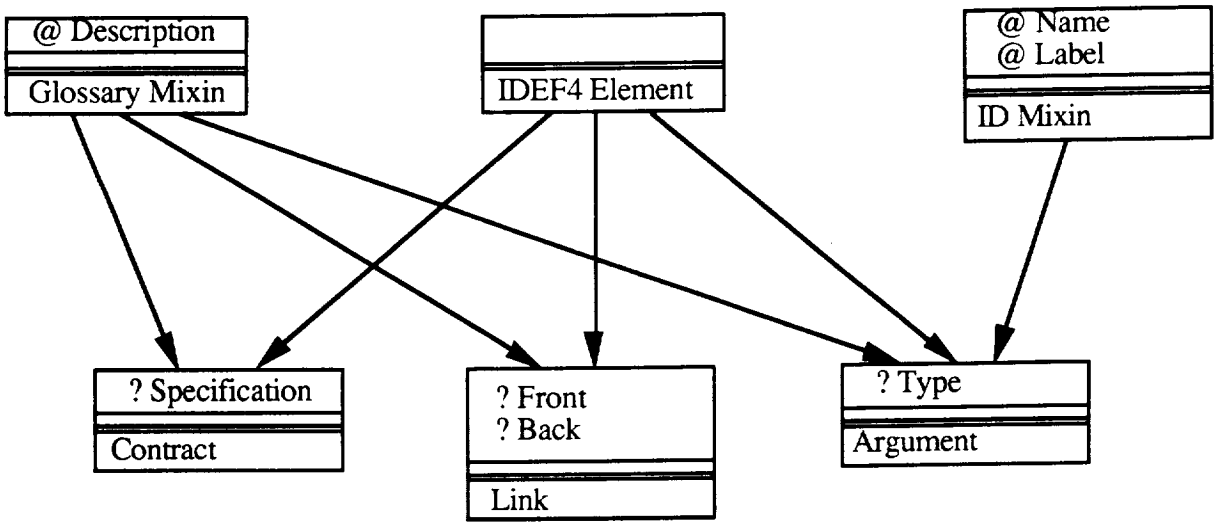


Figure 31. IDEF4 Basic Object Class Inheritance Diagram 2

Figures 30 and 31 show the inheritance relationships that exist between the basic IDEF4 class structures. The relationships in these two diagrams are pretty trivial: every class is an IDEF4 element and every class has a glossary description associated with it. Probably the most important thing to notice is that the Feature Use in Class class in Figure 30 and the Contract and Link classes in Figure 31 do not inherit the ID Mixin. This implies that uniqueness will be determined by other means and that instances of these classes will have to be accessed through other objects that refer to them. The type diagrams above show how these classes will be related to other objects within an IDEF4 design.

Finally, Figure 32 displays the inheritance structure of the IDEF4 diagram objects. The Diagram object inherits from IDEF4 Element, Glossary Mixin, and ID Mixin. Since every diagram class inherits from this Diagram class, every diagram will have a name and a glossary entry associated with it. Also note that the Class Inheritance and the Type Diagram classes inherit from the Class Grouping class. This mixin class provides the feature necessary to maintain a list of classes within the class instance. This list will represent the classes that are to appear in that particular diagram instance.

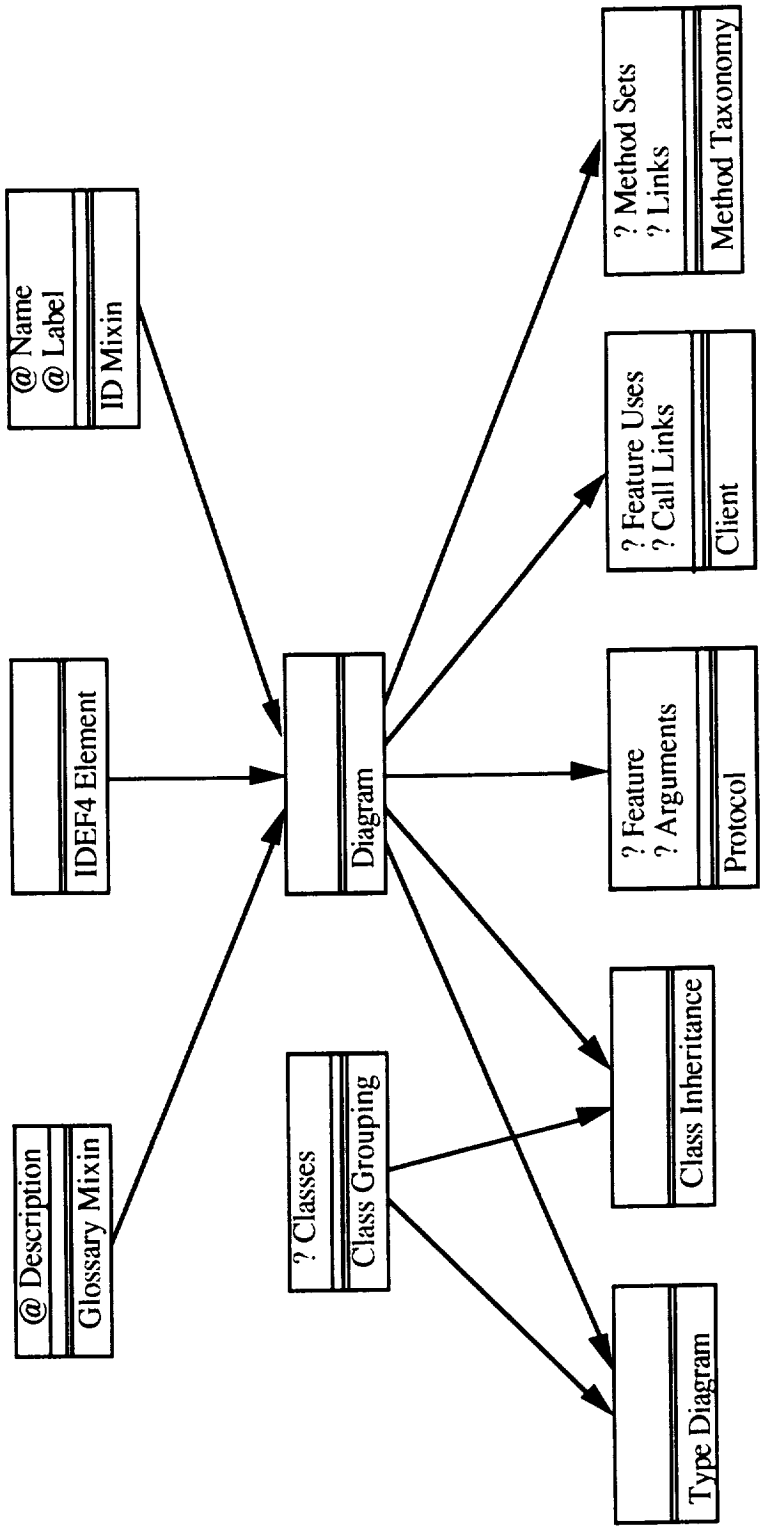


Figure 32. IDEF4 Diagram Objects Class Inheritance Diagram

4.2.2 IDEF4 User Interface and Constraint Enforcement

At this point, the focus of the IDEF4 design description shifts from the objects to be manipulated by the system to the processes that are performed to actually manipulate those objects. Appropriately, IDEF3 process flow descriptions will be used to relate these processes. It is hoped that these process descriptions will give the reader a feel for what it will be like to use the IDEF4 tool. To assist in achieving this goal, some UOBs will have Referents attached to them that point to representative screens. These screens will be drawings of the screen that the user will encounter at that point in the development process. The discussion of the IDEF3 descriptions will indicate any constraints that must be satisfied by a particular process.

Since the IDEF3 methodology is in its infancy, development strategies for producing process descriptions are non existent. In defining the specifications for the prototypes, two different approaches of process development were taken so that understanding of how well IDEF3 works in certain situations could be gained. The process description of the IDEF3 tool was developed on a high level with considerable textual elaboration. However, the IDEF4 tool process description was developed on a lower, more specific level. As a result, there is very little textual descriptions to coincide with the process descriptions. Hopefully, both levels of description are sufficient for our purpose, but comments on reader preferences would certainly be appreciated.

Because of the more detailed approach mentioned above, the IDEF3 description of IDEF4 has resulted in a extremely large number of diagrams. In an attempt to simplify the tracking of the decompositions of the UOBs, the diagrams defining the decomposition for a UOB has been placed as close to the diagram that contains the UOB being decomposed. The List of Figures may also prove helpful in finding the various decompositions.

NOTES:

(1) In these IDEF3 descriptions, the numbering convention of Units of Behavior have been changed. Because every UOB in these descriptions have only one decomposition, if any, the decomposition number after the 'v' in the UOBs has been dropped. For example, if a UOB, 1, had a decomposition with a UOB numbered 3 in the decomposition, that UOBs node

number would have been 1.v1.3. Instead, in the following diagrams, the node number would be 1.v.3. Also, due to the diagram size, the prefix of the UOBs is being removed and placed in the top left corner of the diagram. This notation will simplify the numbering scheme of these diagrams as the number of decompositions increases.

(2) The UOBs that have shadowed boxes are UOBs that have decompositions defined for them.

(3) A special character, ¥, is placed in the upper left corner of a UOB to indicate that that particular UOB represents an explicit user action. UOBs without that character represent general procedures of processes performed by the tool.

(4) Throughout the process descriptions, certain UOBs that indicate user actions have referents attached to them labelled "See IDEF4 Screen #". Sketches of the proposed screens for the IDEF4 tool have been created and can be found at the end of this section. These referents are refering to these screens so that the reader can get a feel for what the system will look like when performing certain operations.

There is one aspect of the IDEF4 tool that was not described in the process descriptions because of the added complexity it would add to the descriptions. During the execution of any of the process supported by the tool, it is possible for the user to abort out of the process. The effect of this abort is to return to the state that existed before the aborted process was started.

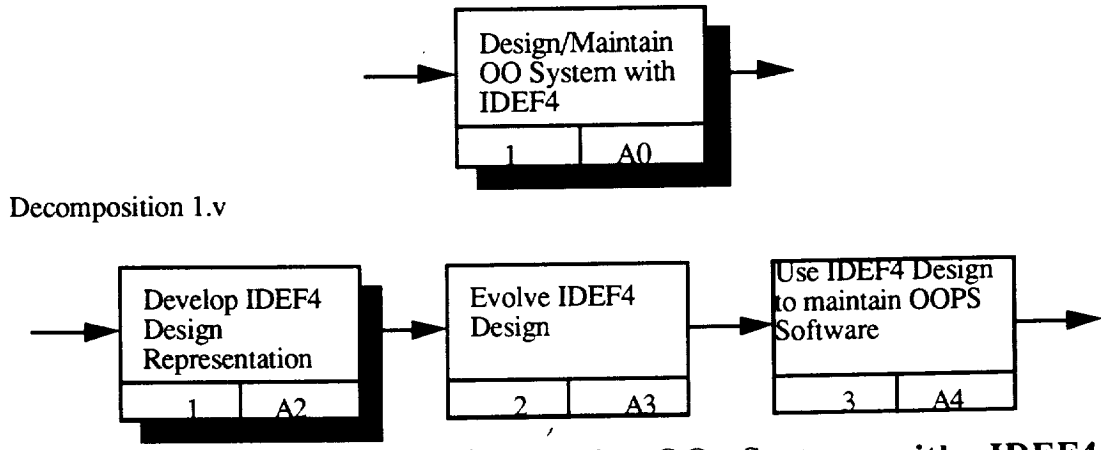


Figure 33. Design/Maintain OO System with IDEF4 Scenario

Figure 33 contains the first two levels of the IDEF4 tool process description. The process of designing an object oriented system is broken down into the three processes defined at the bottom of the figure. Though these three processes are very similar in the operations that are performed, the distinction is made to indicate that different aspects of the design will be accessed during these processes and that the processes will be performed by different people. In this description, the focus will be on the Develop IDEF4 Design Representation Process.

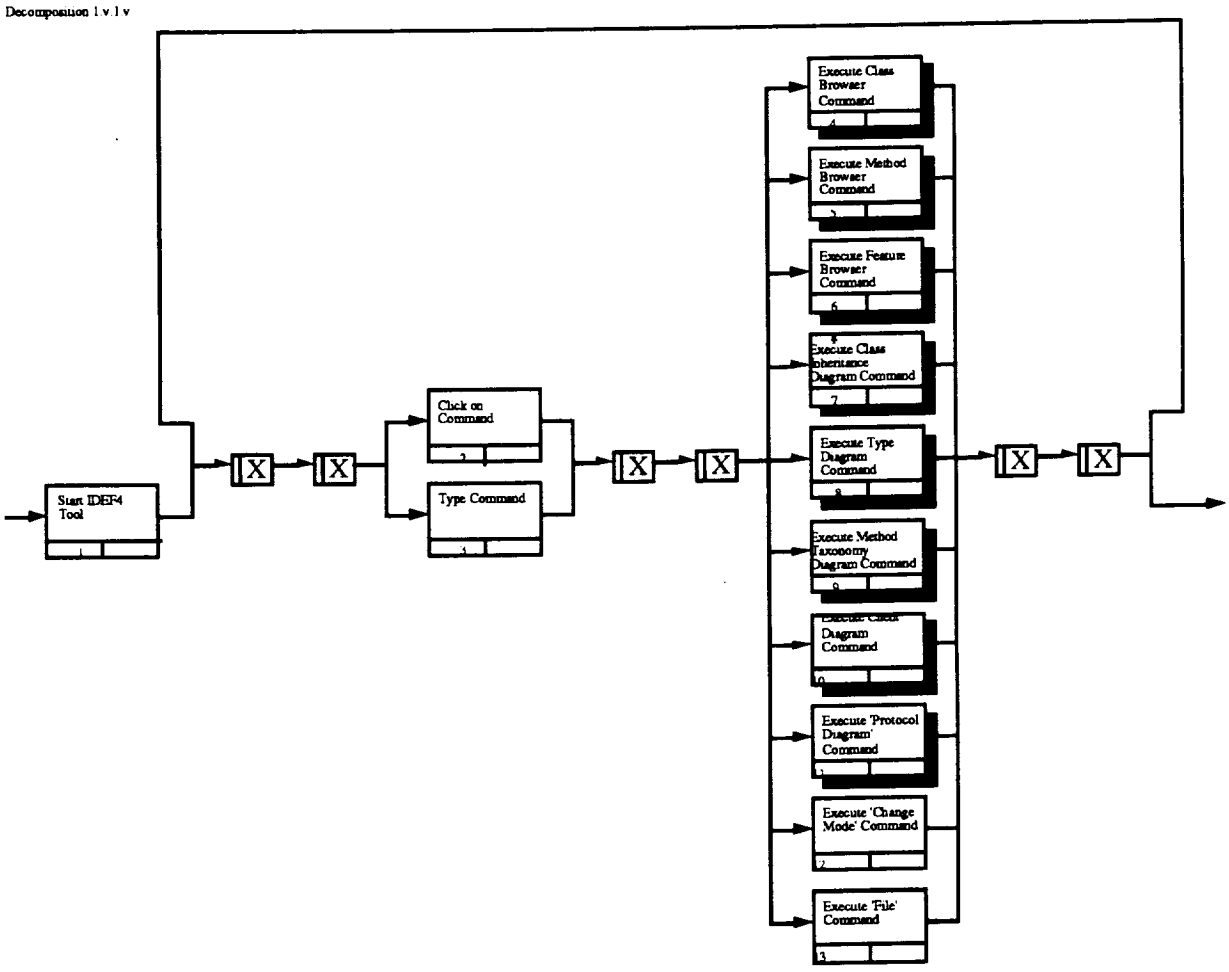


Figure 34. Decomposition of Develop IDEF4 Design Representation

Figure 34 shows a high level view of the execution of the IDEF4 tool. After loading the IDEF4 tool, the user has the option of either typing a command at the command line or selecting a command from the command menu. The column of processes on the right of the

diagram represent the different modes of operation that the IDEF4 tool can operate under. Also notice that the command execution phase of the process is an iterative process.

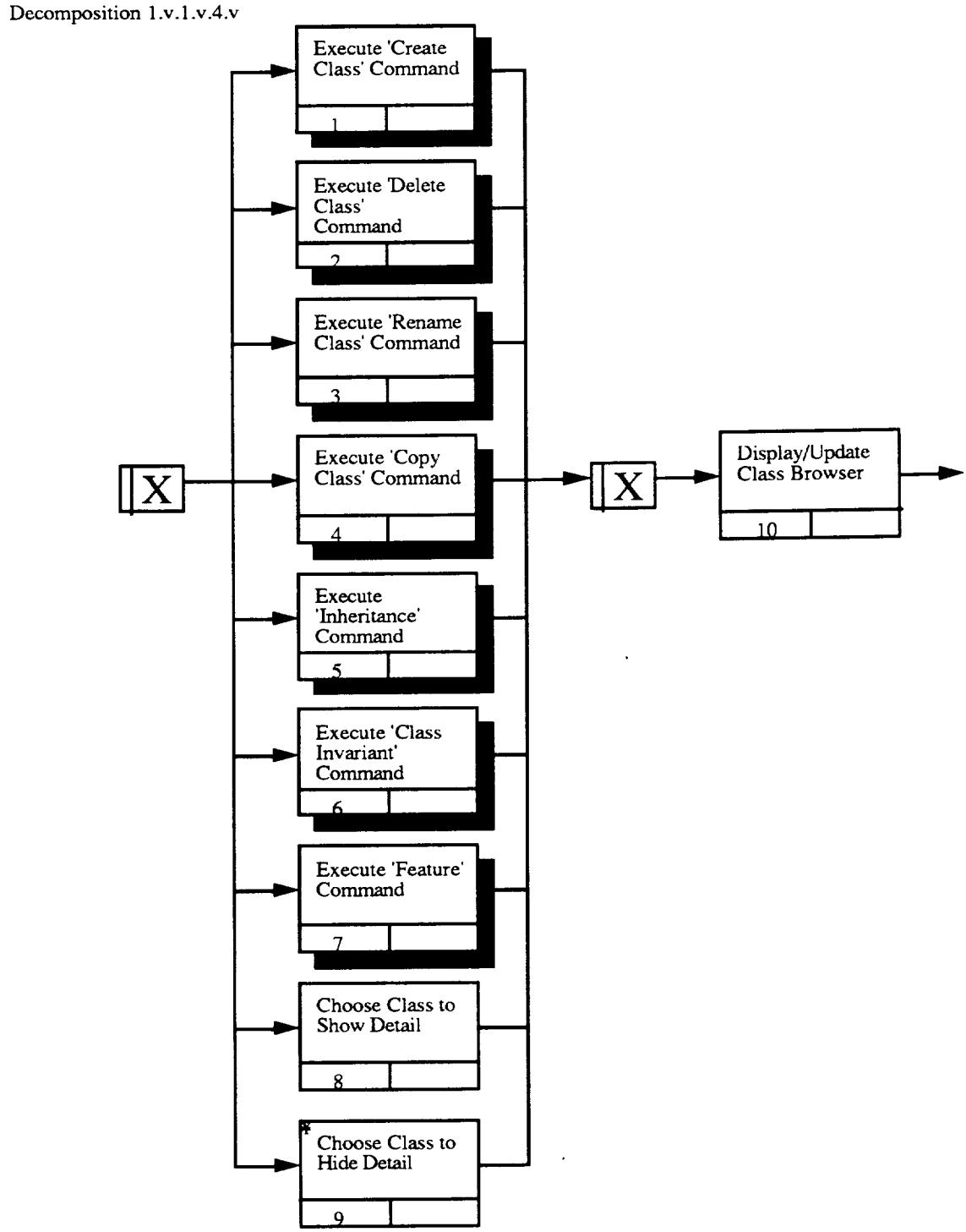


Figure 35. Decomposition of Process Class Diagram Command

Figure 35 defines the various commands that are available for execution within the Class Browser Mode of the tool. Figures 36 through 43 define these commands in greater detail. It should be noted that some of this functionality will overlap with other modes of operations. Also, certain constraints exist that must be satisfied for the operations to be performed correctly.

Constraints:

- When adding features to a class, features that have already been "used" by other classes will not be available. The only way a class can use a feature that is used by another class is through an inhertance link.
- When a class is deleted, classes that inherit from the deleted class should have the inherited features removed from them.
- When an inheritance link is delete, any features that were inherited as a result of that link should be removed from the subclass and any subsequent subclasses.
- When specifying inheritance links, the tool should ensure that no circular links are defined. Otherwise, a class could be a parent class of itself.
- If the definition of an inheritance will result in a conflict of two features, the tool should automatically redefine the feature.

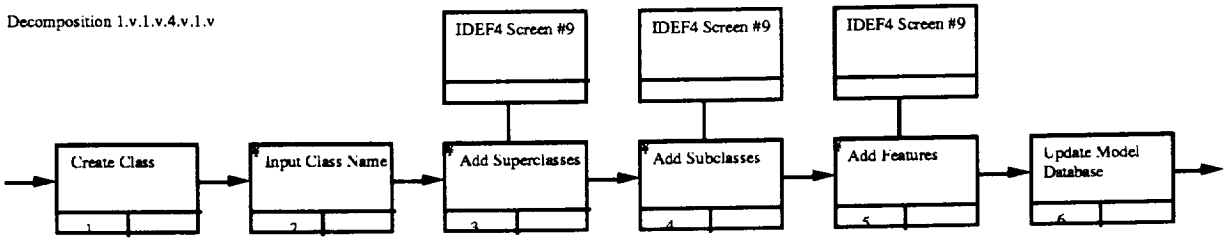


Figure 36. Decomposition of Execute Create Class Command

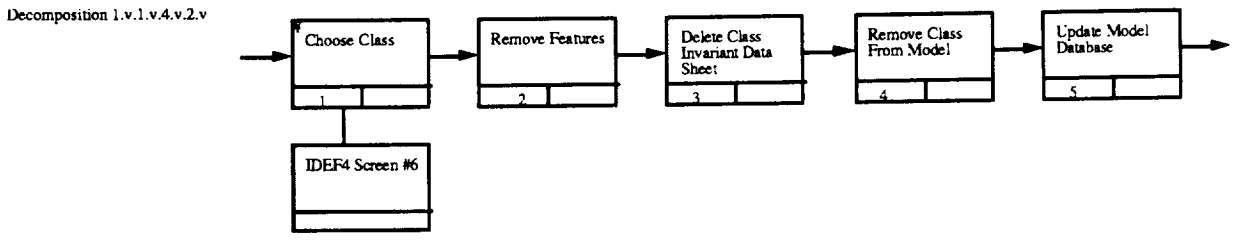


Figure 37. Decomposition of Execute Delete Class Command

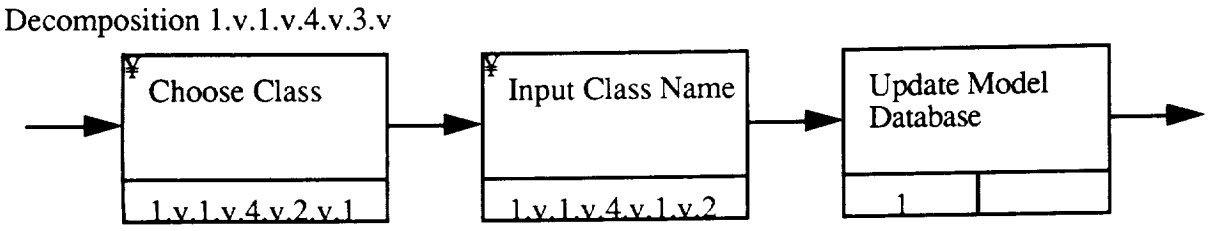


Figure 38. Decomposition of Execute Rename Class Command

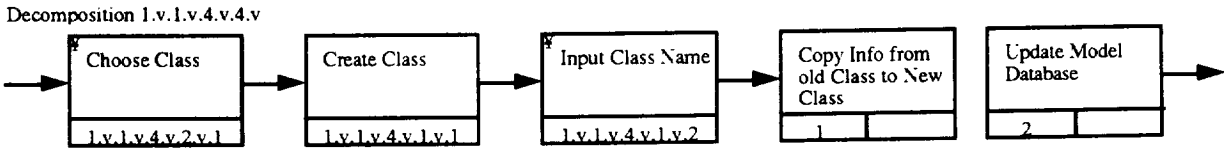


Figure 39. Decomposition of Execute Copy Class Command

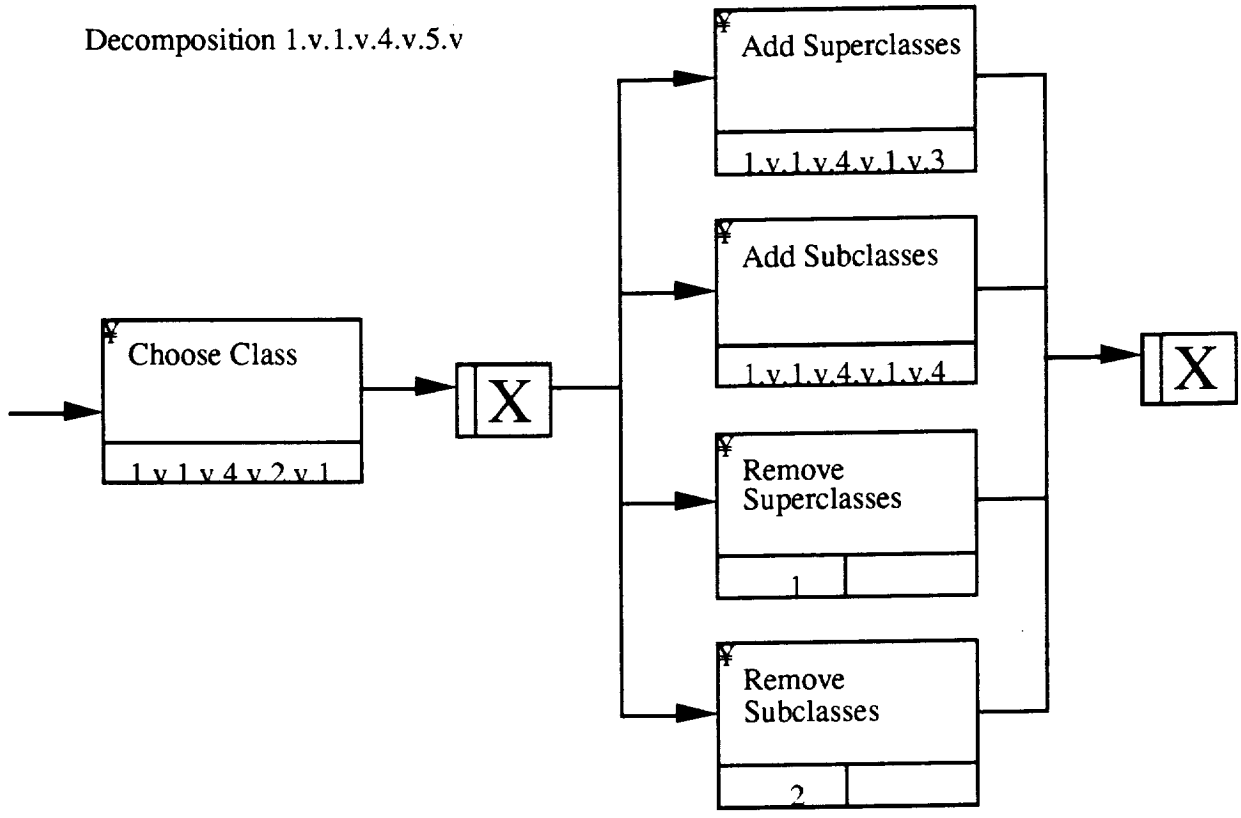


Figure 40. Decomposition of Execute Inheritance Command

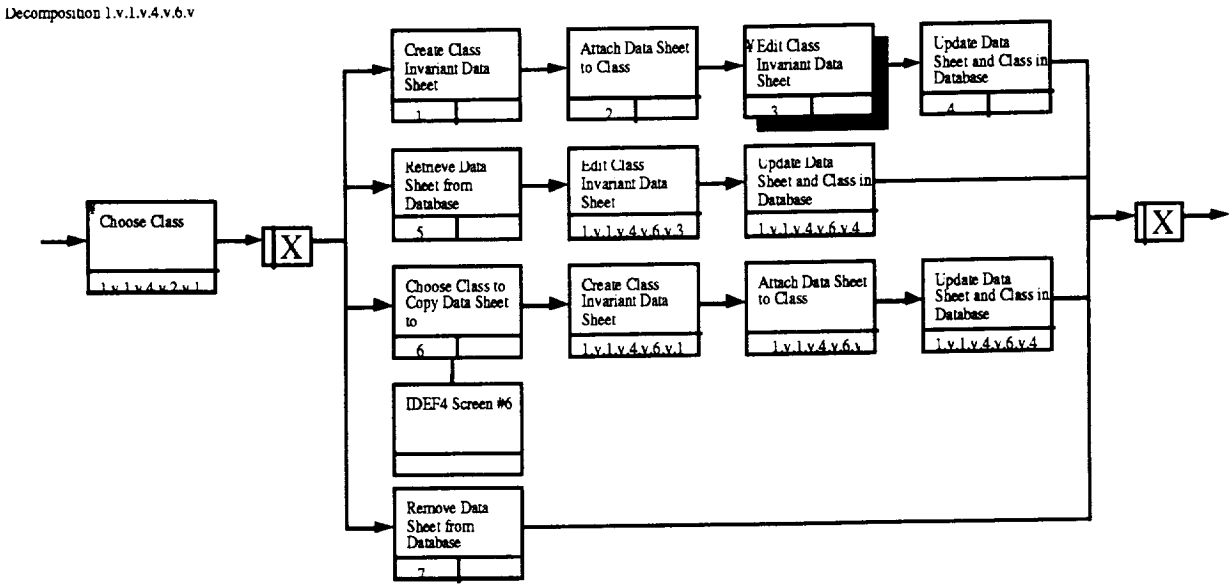


Figure 41. Decomposition of Execute Class Invariance Command

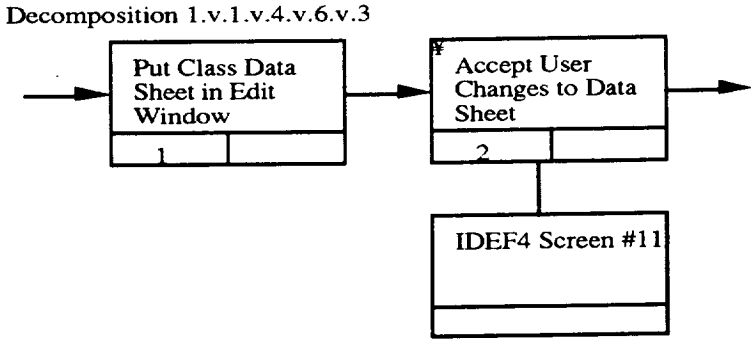


Figure 42. Decomposition of Edit Class Invariance Data Sheet

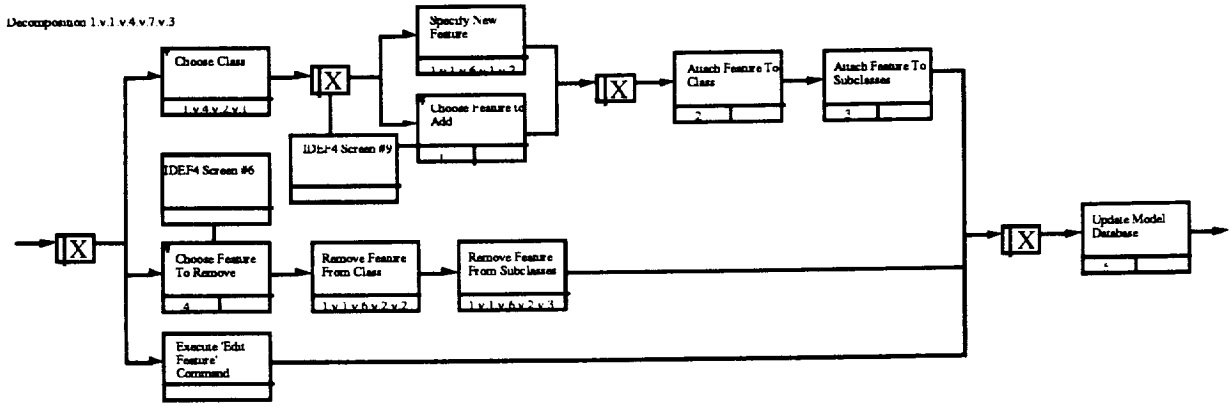


Figure 43. Decomposition of Execute Feature Command

Decomposition 1.v.1.v.5.v

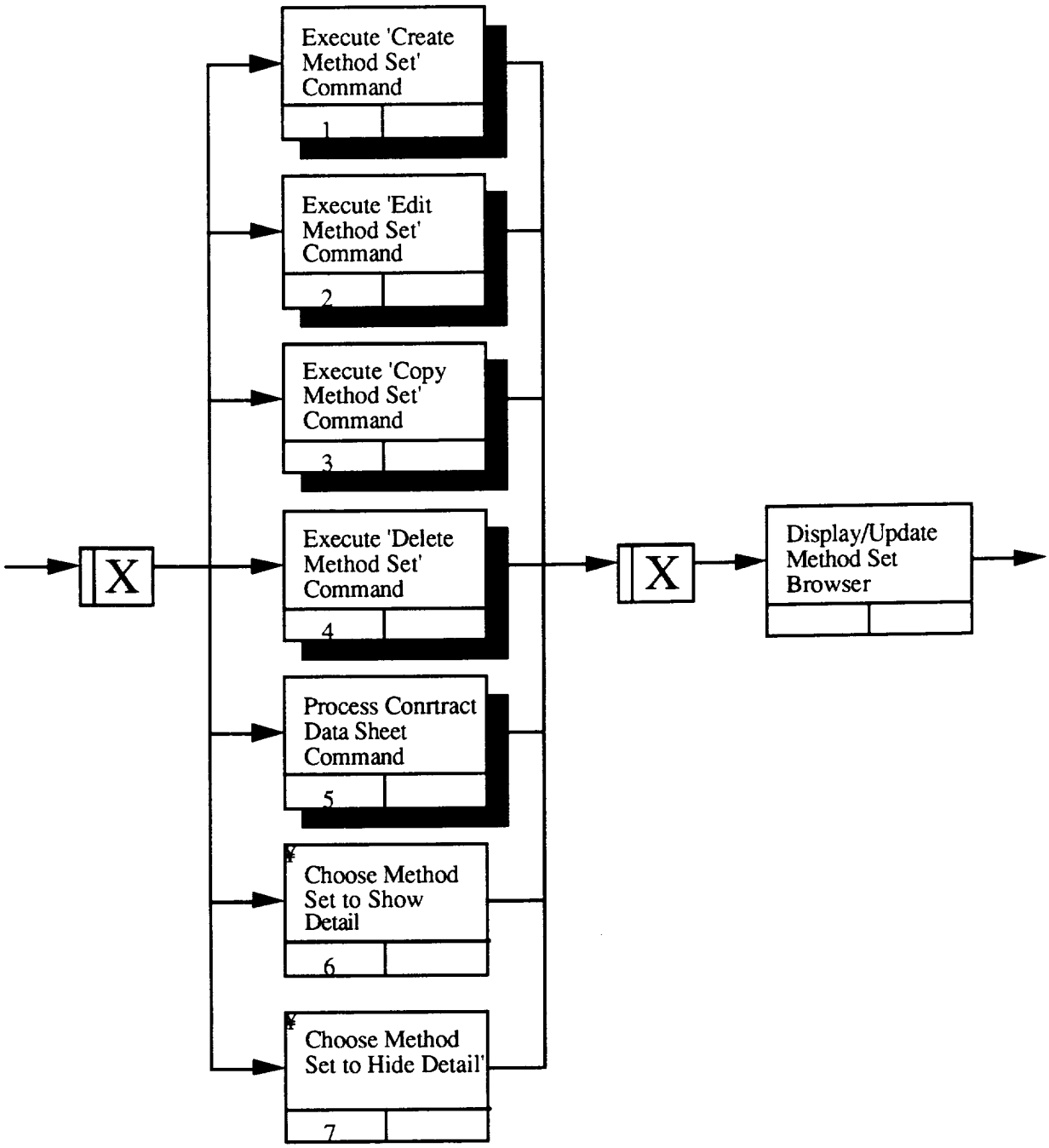


Figure 44. Decomposition of Execute Method Browser Command

Figure 44 outlines the commands that can be executed from the Method Set Browser. Figures 45 through 50 describe these operations in greater detail.

Constraints:

- A Method Set can have only one contract data sheet, though many contacts may be specified. The Create Contract Data Sheet Command must ensure that the Method Set does not have an existing data sheet.
- When a Method Set is deleted, any class/feature pairs that specify a contract for the deleted method set should be automatically updated.

Decomposition 1.v.1.v.5.v.1.v

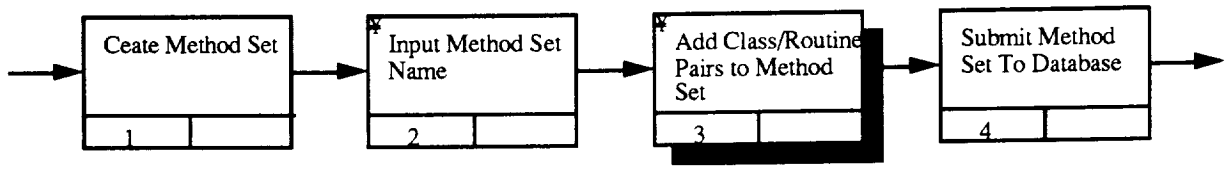


Figure 45. Decomposition of Execute Create Method Set Command

Decomposition 1.v.1.v.5.v.1.v.3.v

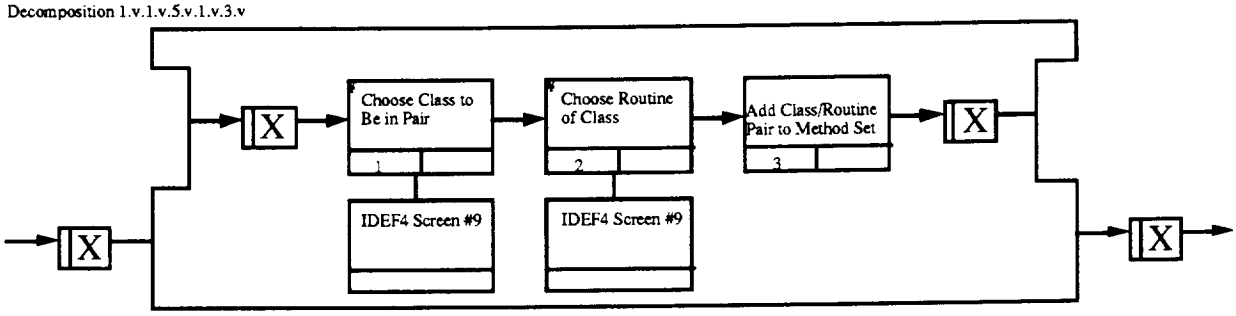


Figure 46. Decomposition of Add Class/Routine Pairs to Method Set

Decomposition 1.v.1.v.5.v.2.v

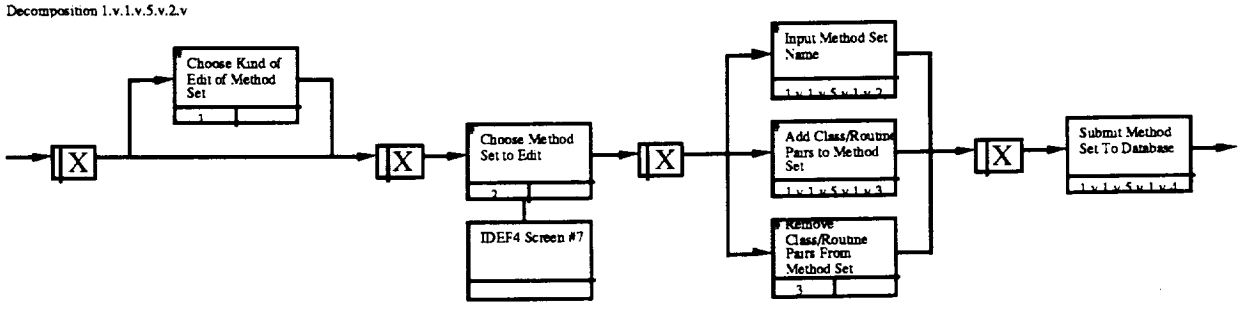


Figure 47. Decomposition of Execute Edit Method Set Command

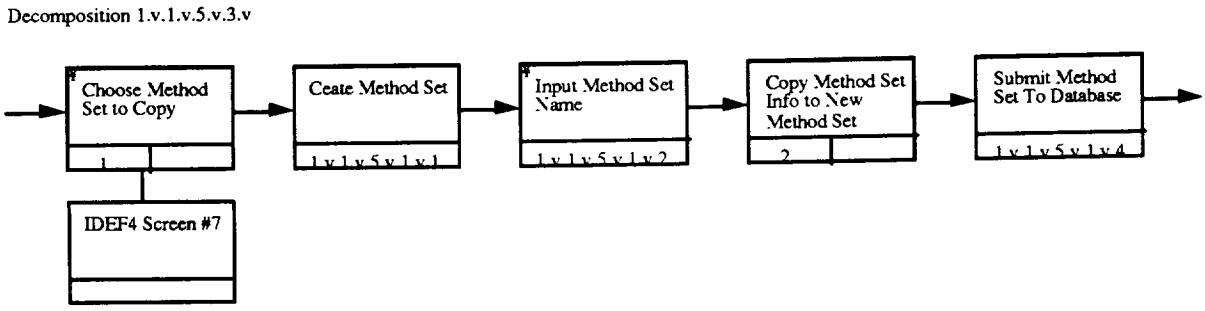


Figure 48. Decomposition of Execute Copy Method Set Command

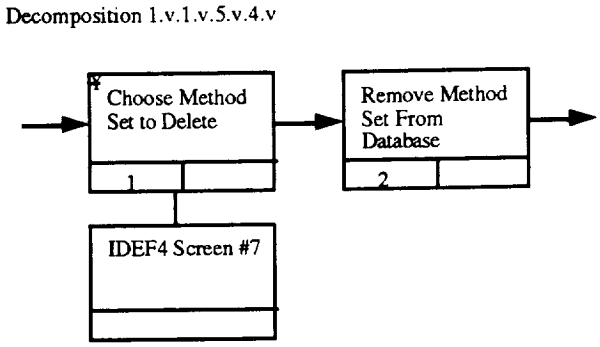


Figure 49. Decomposition of Execute Delete Method Set Command

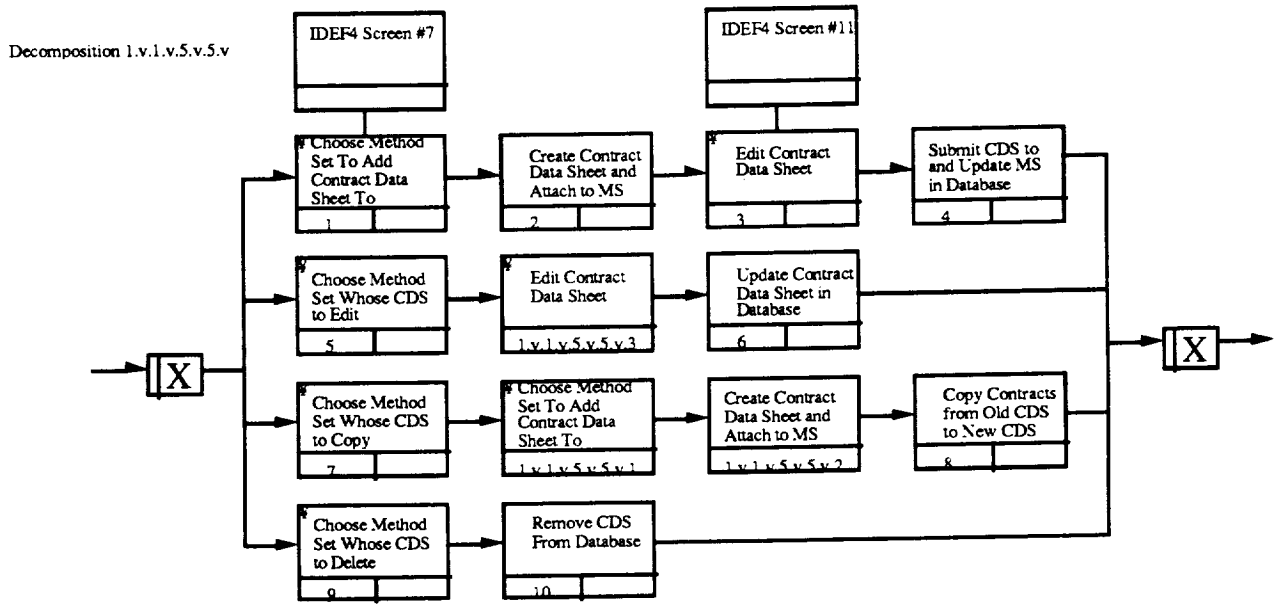


Figure 50. Decomposition of Process Contract Data Sheet Command

Decomposition 1.v.1.v.6.v

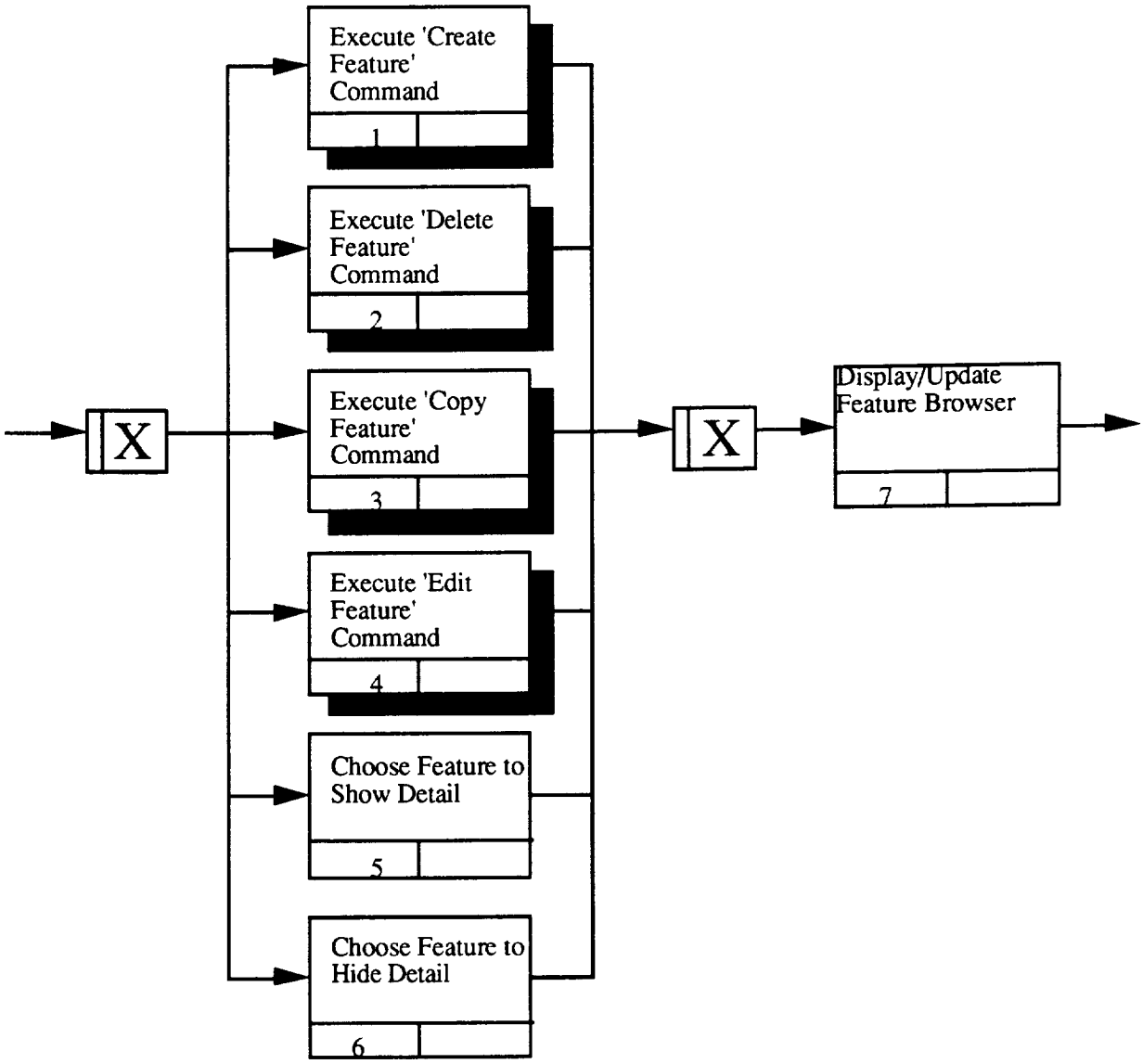


Figure 51. Decomposition of Execute Feature Browser Command

Figure 51 defines the commands that would be available in the Feature Pool Browser. Figures 52 through 56 describe these operations in greater detail.

Constraints:

- When a feature is deleted, all classes that use the feature should have the feature removed.
- When a feature is deleted from a specific class, that class, and any classes that inherit from that class should be updated to reflect the removal of the feature.
- When a feature is modified, all classes that have the feature, either by direct ownership or through inheritance, should be updated to reflect the modification.

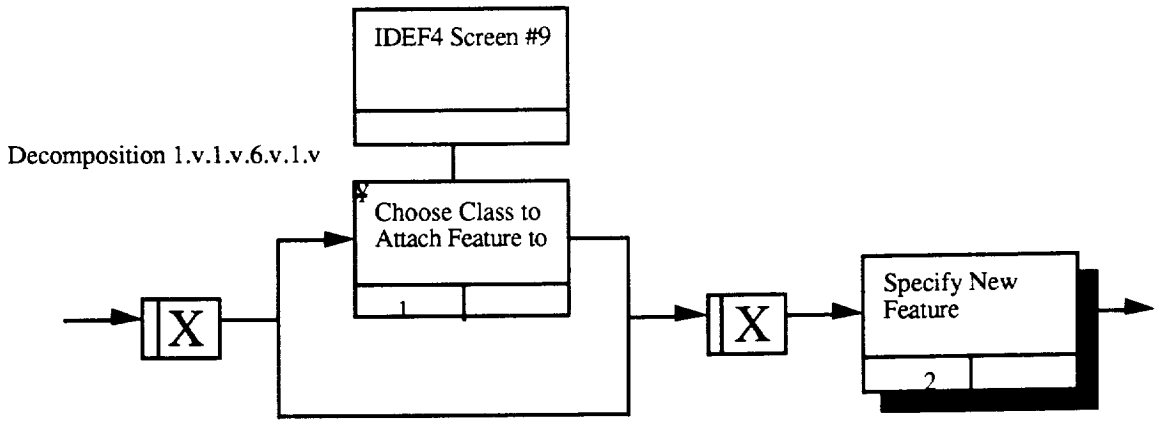


Figure 52. Decomposition of Execute Create Feature Command

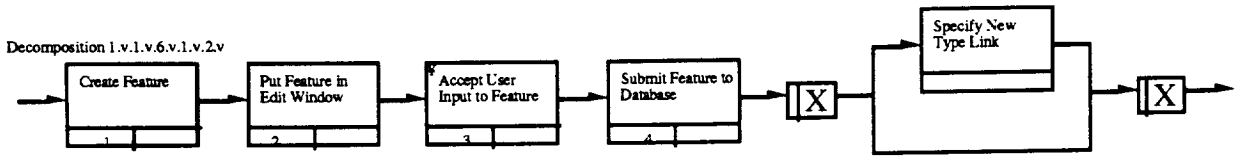


Figure 53. Decomposition of Specify New Feature

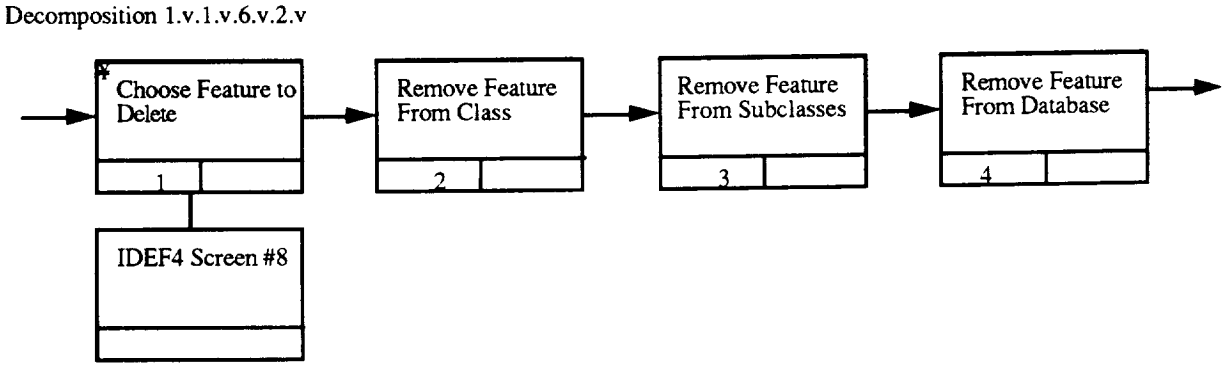


Figure 54. Decomposition of Execute Delete Feature Command

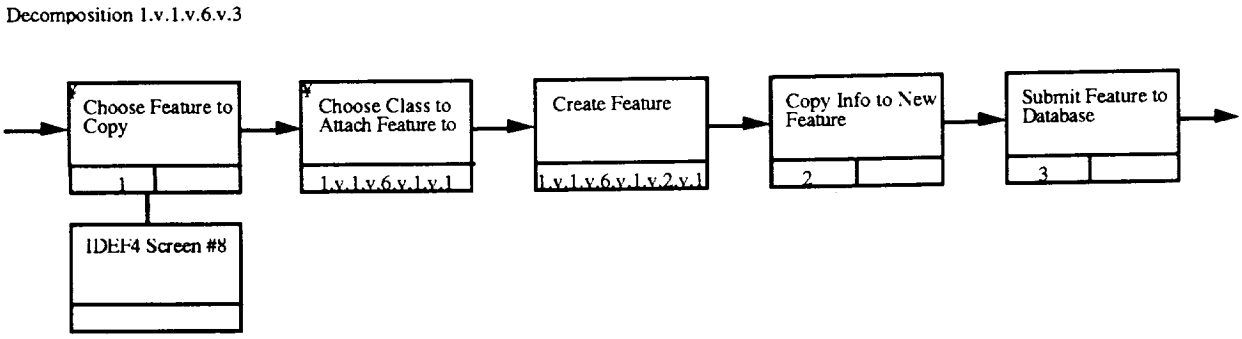


Figure 55. Decomposition of Execute Copy Feature Command

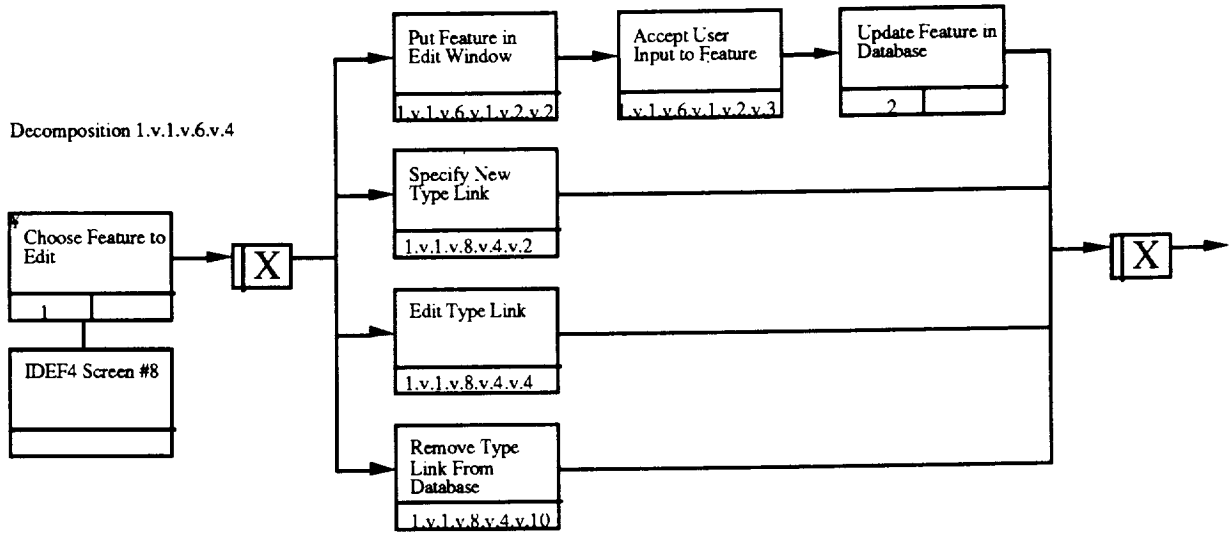


Figure 56. Decomposition of Execute Edit Feature Command

Decomposition 1.v.1.v.7.v

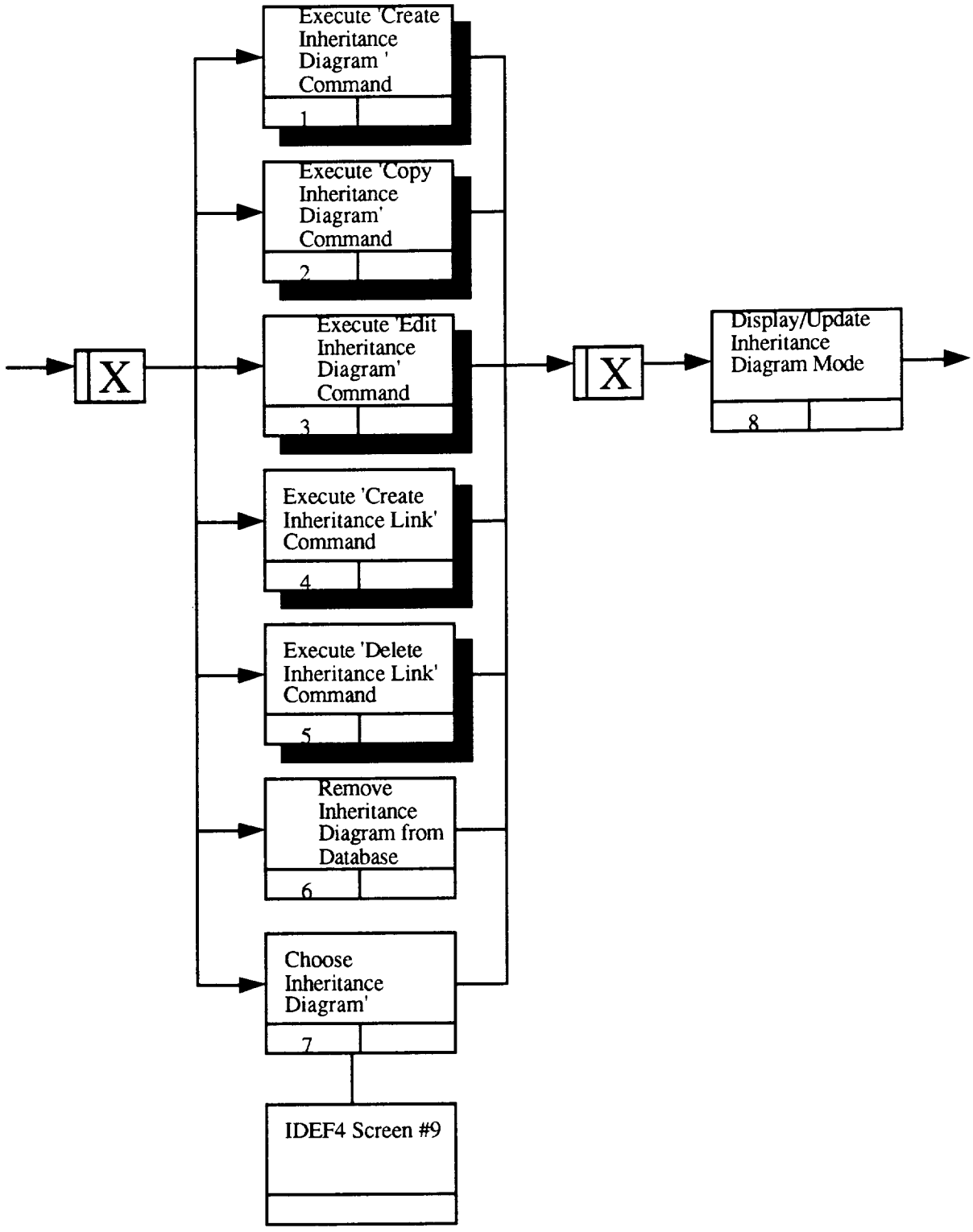


Figure 57. Decomposition of Execute Class Inheritance Diagram Command

Figure 57 defines the operations that are available within the Class Inheritance Diagram Facility. This diagram facility can be used in two ways. The first is to use the diagram as a viewing facility of class relationships that are defined in the Class Browser. The second way is to use the diagram facility as the means to create classes and to define relationships between those classes. Over the course of a system design it is likely that the designer will use both methods.

Figures 58 through 63 define the Inheritance diagram operations in greater detail. The constraints associated with these operations are identical to the constraints that were specified for the Class Browser procedures. The reason for this is that there is a great deal of functional duplication between the two modes of operation.

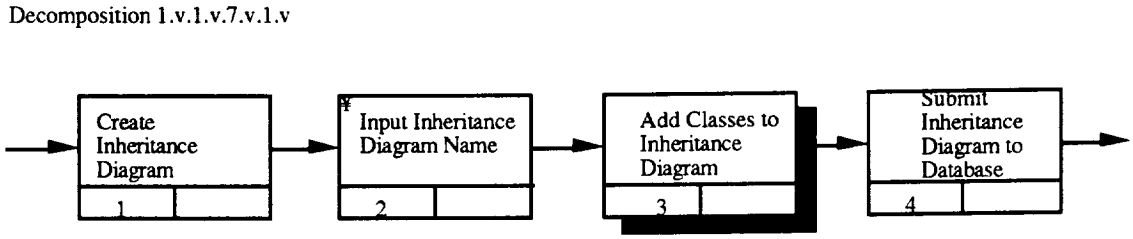


Figure 58. Decomposition of Execute Create Inheritance Diagram Command

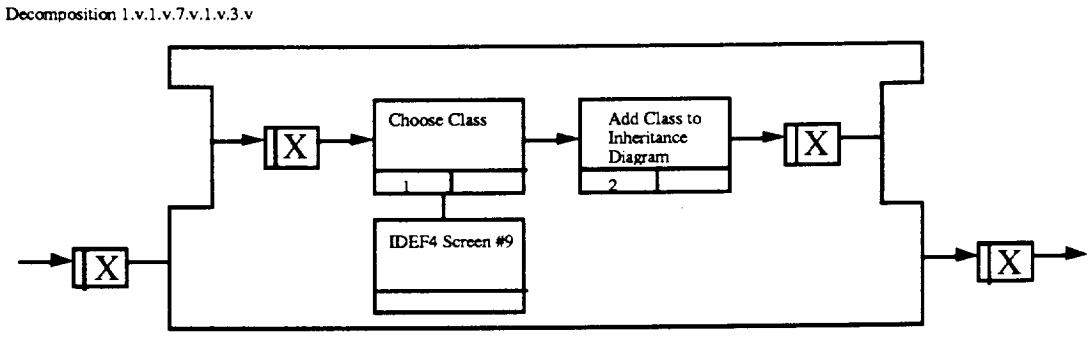


Figure 59. Decomposition of Add Classes to Inheritance Diagram

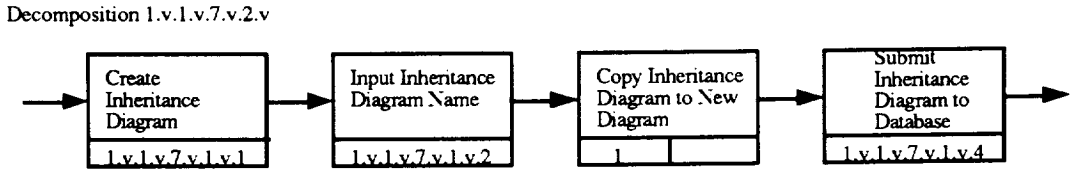


Figure 60. Decomposition of Execute Copy Inheritance Diagram Command

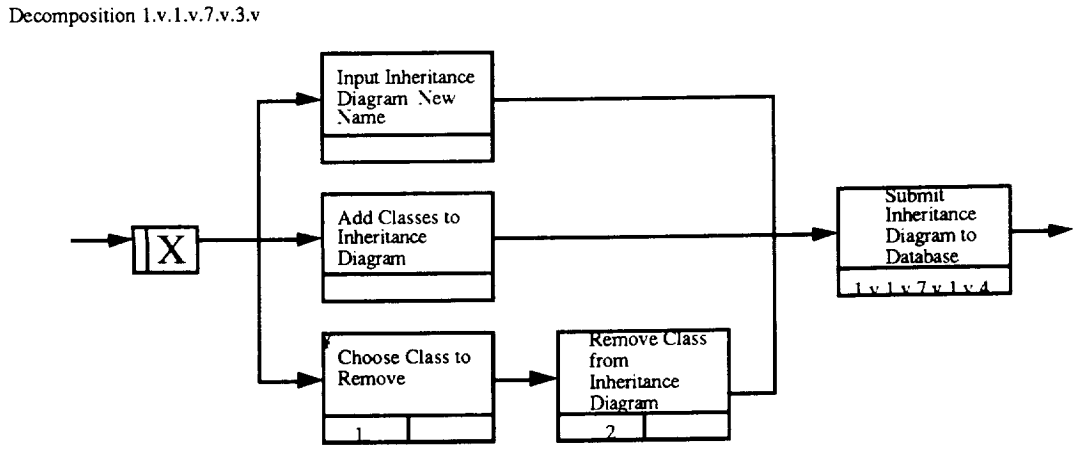


Figure 61. Decomposition of Execute Edit Inheritance Diagram Command

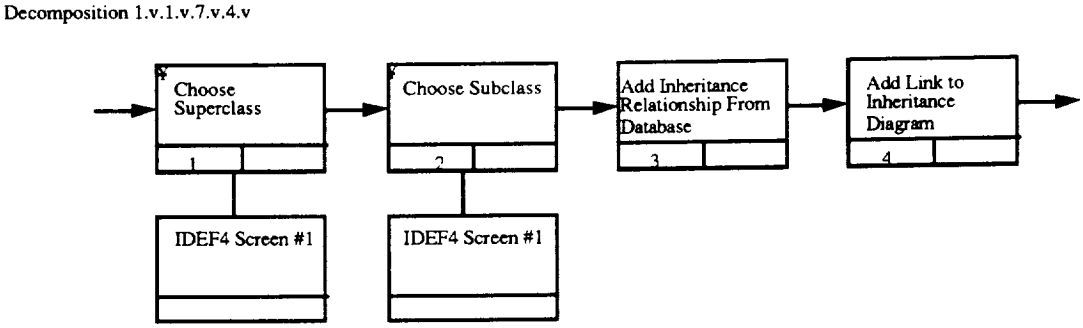


Figure 62. Decomposition of Execute Create Inheritance Link Command

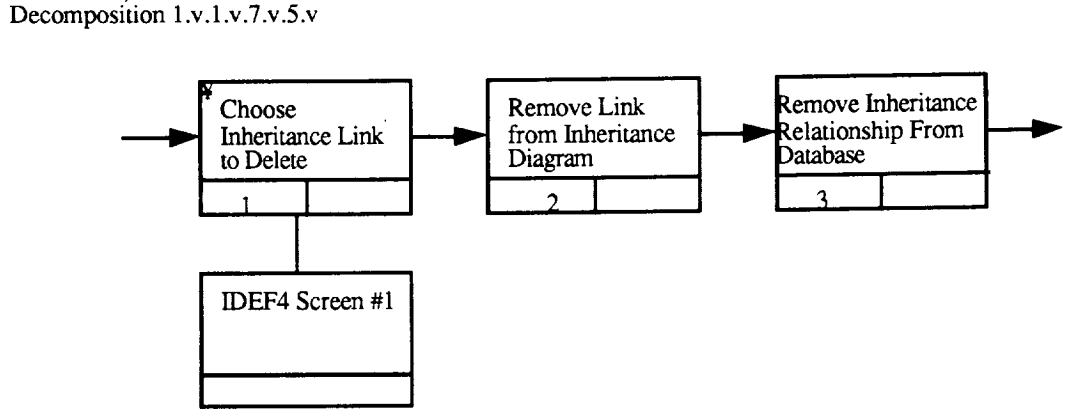


Figure 63. Decomposition of Execute Delete Inheritance Link Command

Decomposition 1.v.1.v.8.v

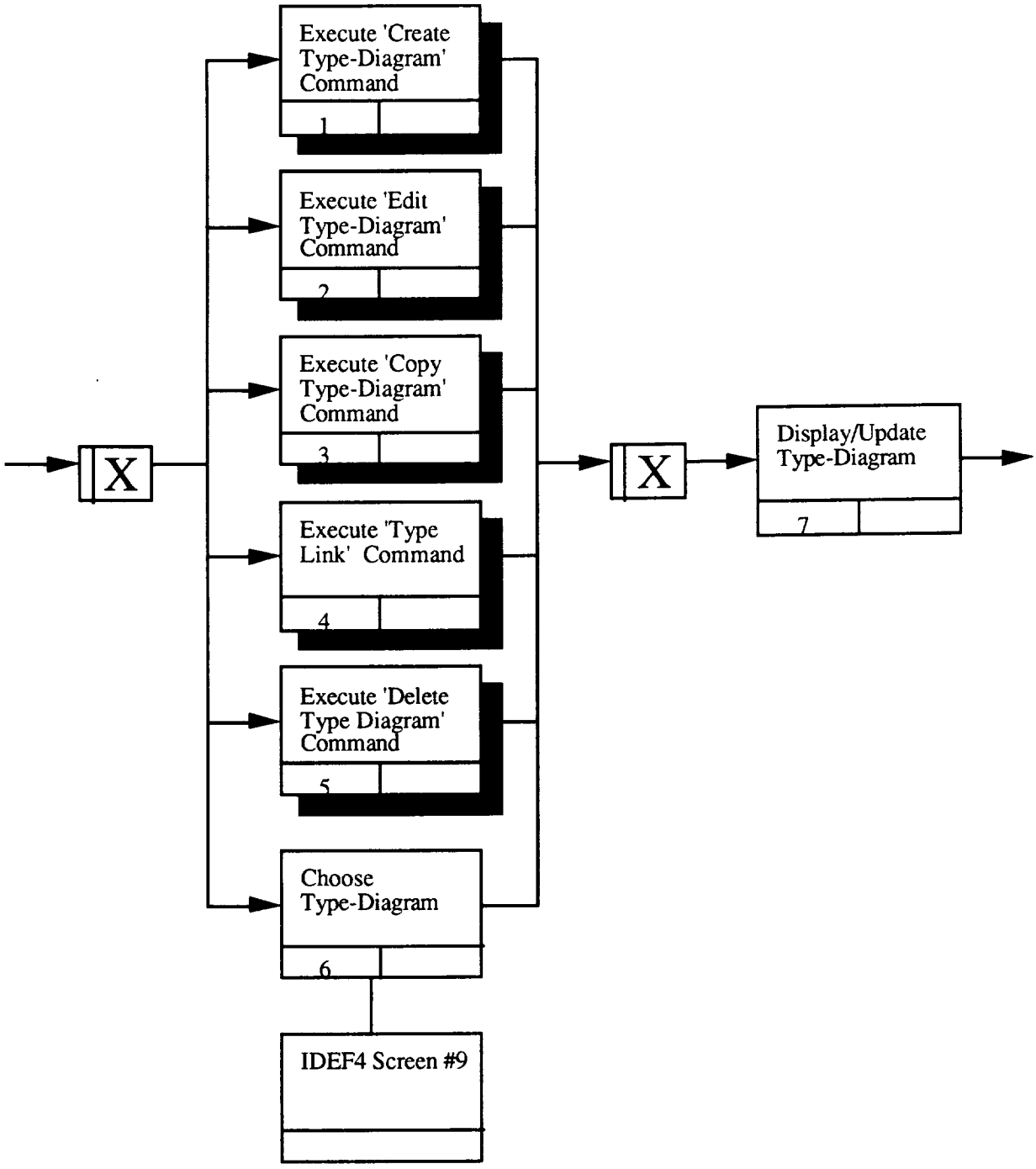


Figure 64. Decomposition of Execute Type Diagram Command

Figure 64 describes the operations available in the Type Diagram Facility. As with the Inheritance diagrams, this diagram facility can

be used in two ways. The first is to use the diagram as a viewing facility of feature type relationships that are defined in the Feature Pool Browser. The second way is to use the diagram facility as the means to create features, assign them to classes, and to define the feature type relationships. Over the course of a system design it is likely that the designer will use both methods.

Figures 65 through 71 describe the Type diagram operations in greater detail.

Constraints:

- When a type link for a feature is deleted, all uses of that feature should have their type links deleted, unless the feature has been redefined in the subclass.
- When a type link for a feature is modified, all uses of that feature should have their type links updated, unless the feature has been redefined in the subclass.
- The deletion of a Type diagram should have no effect on the type links of the class features appearing in that diagram. Only the organizing structure of the diagram will be destroyed.

Decomposition 1.v.1.v.8.v.1.v

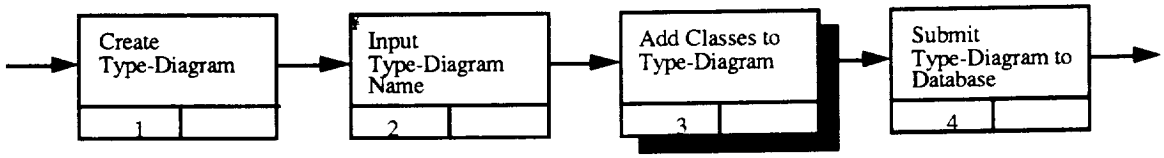


Figure 65. Decomposition of Execute Create Type Diagram Command

Decomposition 1.v.1.v.8.v.1.v.3.v

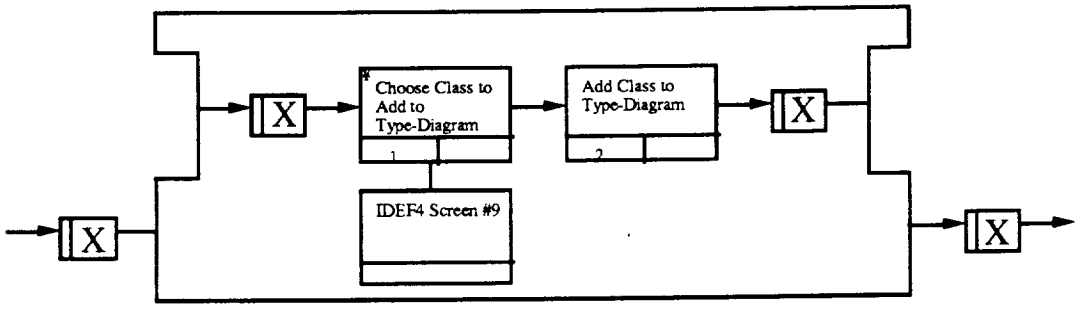


Figure 66. Decomposition of Add Classes to Type Diagram Command

Decomposition 1.v.1.v.8.v.2.v

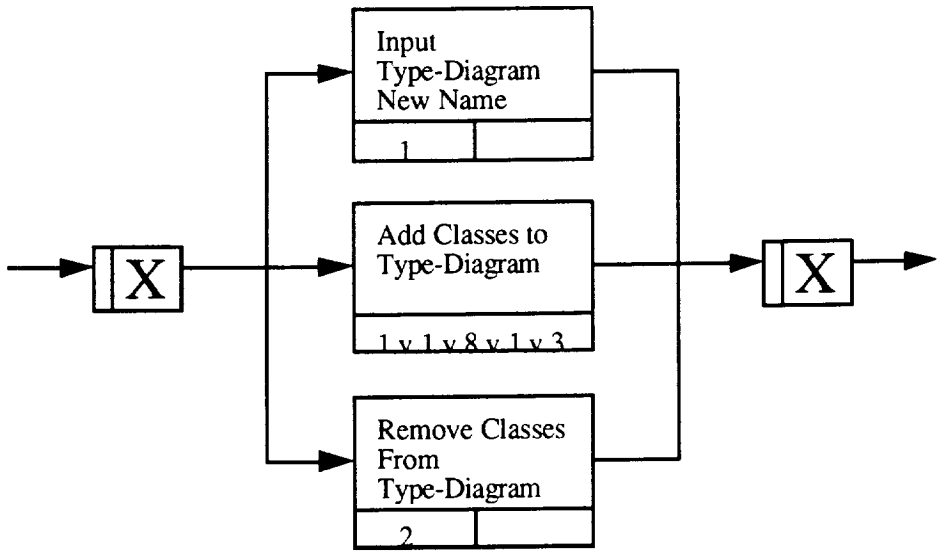


Figure 67. Decomposition of Execute Edit Type Diagram Command

Decomposition 1.v.1.v.8.v.3.v

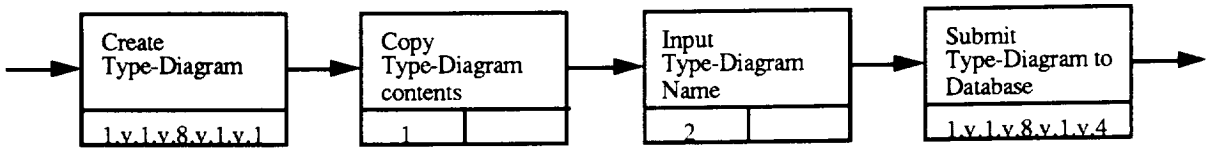


Figure 68. Decomposition of Execute Copy Type Diagram Command

Decomposition 1.v.1.v.8.v.4.v

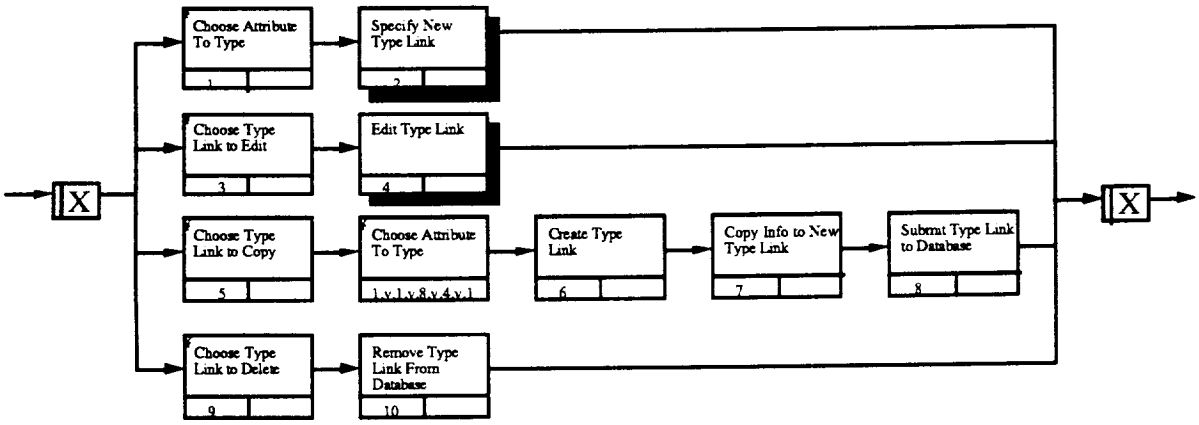


Figure 69. Decomposition of Execute Type Link Command

Decomposition 1.v.1.v.8.v.4.v.2.v

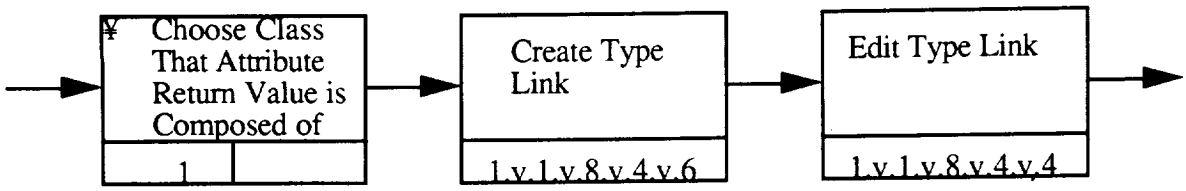


Figure 70. Decomposition of Specify New Type Link

Decomposition 1.v.1.v.8.v.4.v.4.v

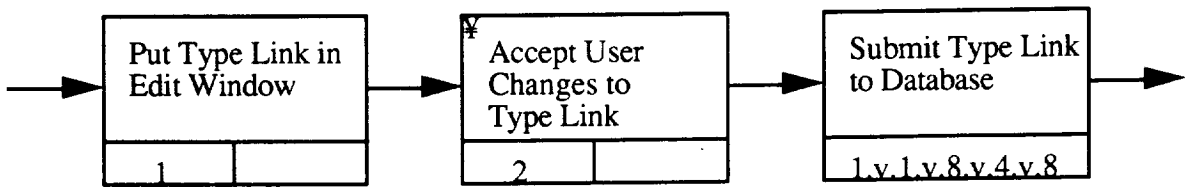


Figure 71. Decomposition of Edit Type Link

Figure 72 defines the operations that would be available in the Method Taxonomy Diagram facility. Figures 73 through 77 provide the details for these operations.

Constraints:

- Deletion of a Method Taxonomy diagram results in all taxonomic links between methods sets being deleted as well.

Decomposition 1.v.1.v.9.v

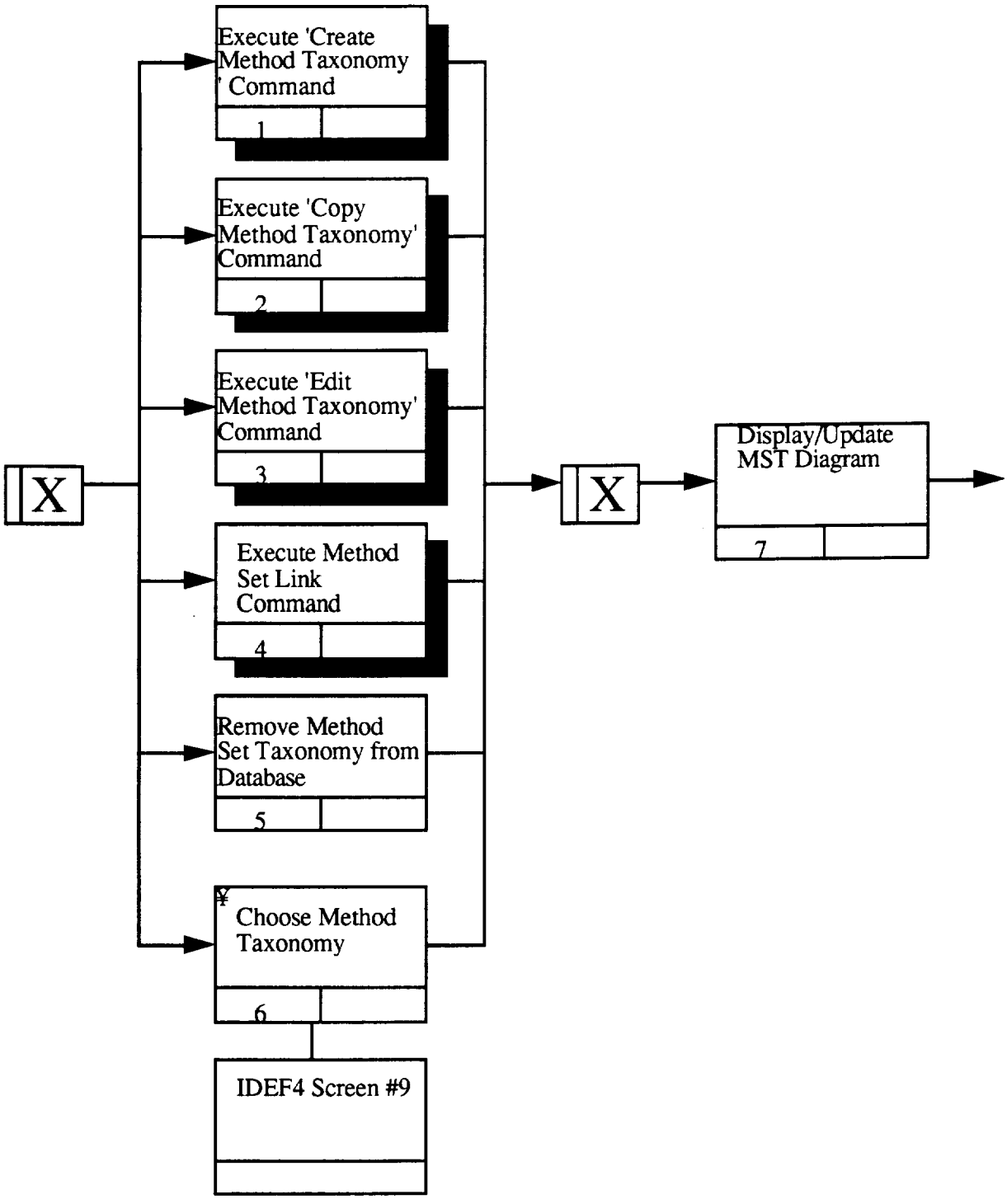


Figure 72. Decomposition of Execute Method Taxonomy Diagram Command

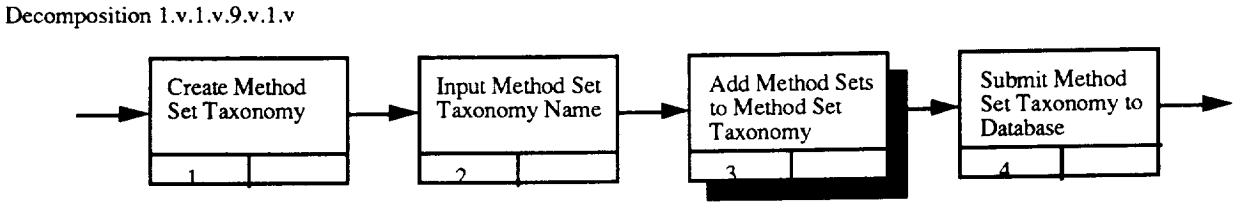


Figure 73. Decomposition of Execute Create Method Taxonomy Command

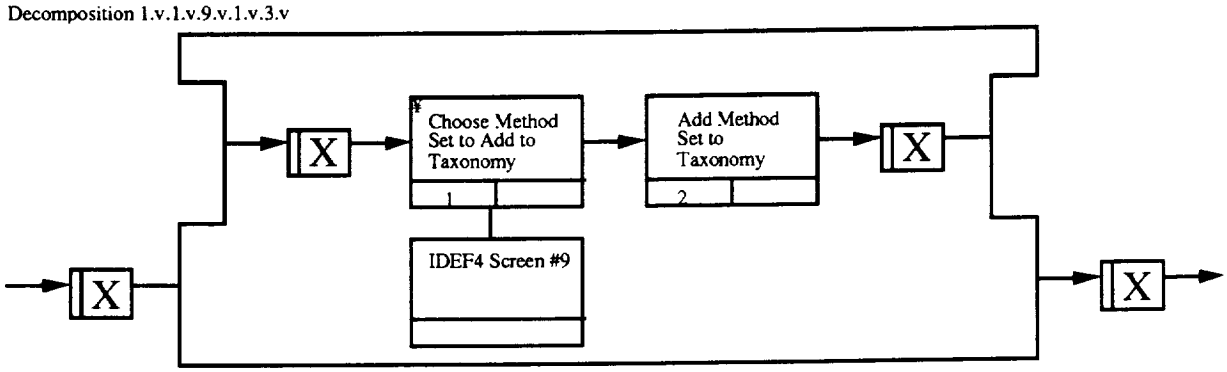


Figure 74. Decomposition of Add Method Sets to Method Set Taxonomy

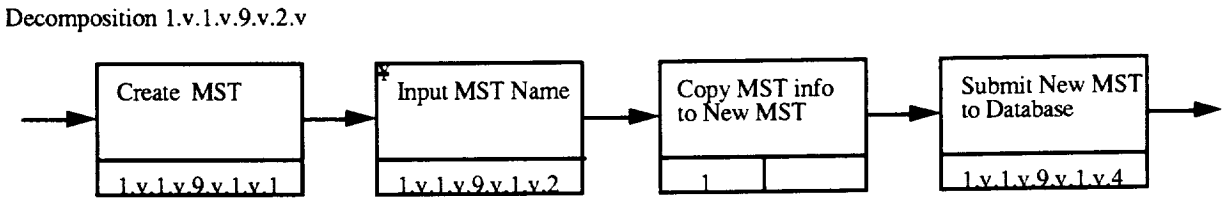


Figure 75. Decomposition of Execute Copy Method Taxonomy Command

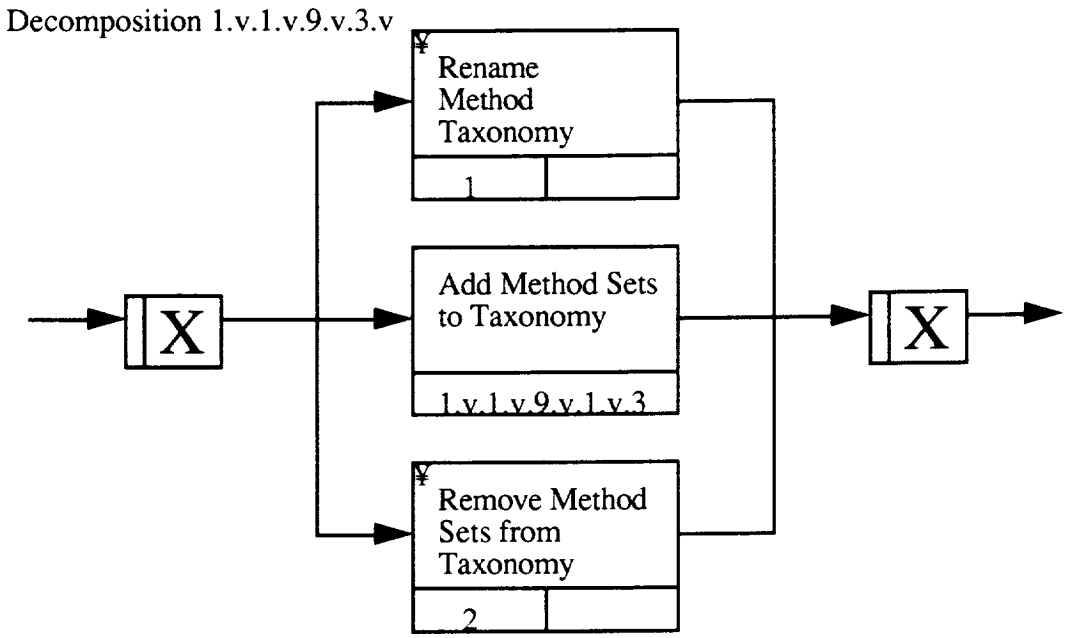


Figure 76. Decomposition of Execute Edit Method Taxonomy Command

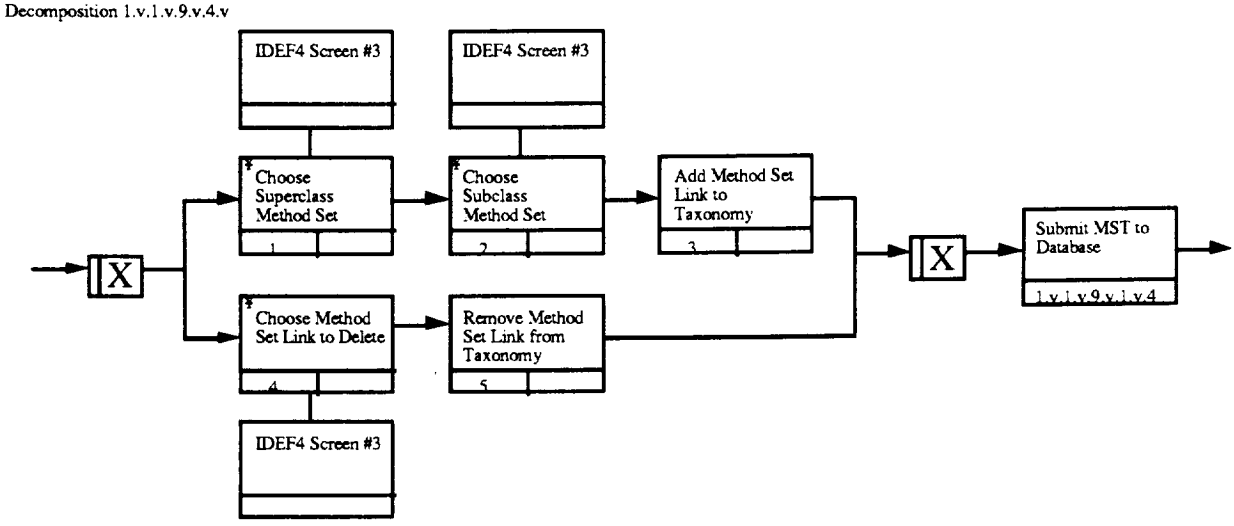


Figure 77. Decomposition of Execute Method Set Link Command

Decomposition 1.v.1.v.10.v

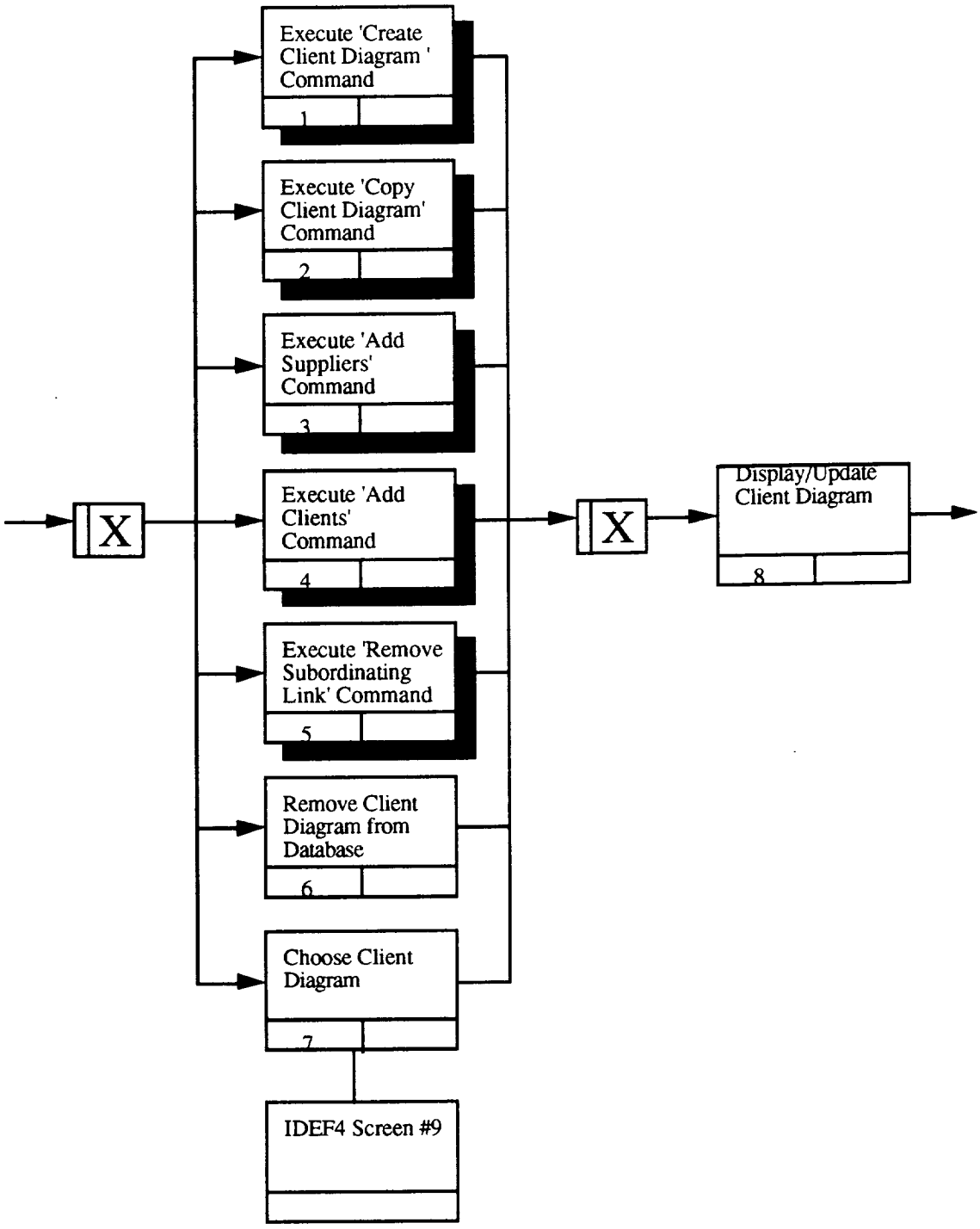


Figure 78. Decomposition of Execute Client Diagram Command

Figure 78 describes the operations that would be available in the Client Diagram facility. Figures 79 through 85 provide more detail on the execution of these procedures.

Constraints:

- Deleteion of a Client Diagram would result in the deletion of all client relationships defined in that diagram.

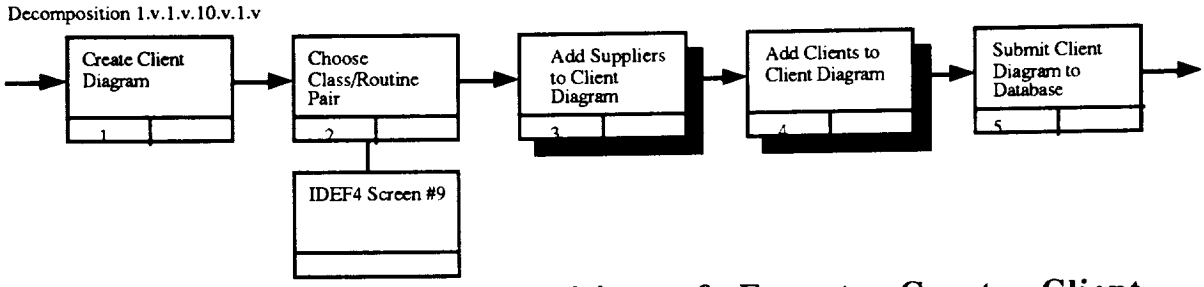


Figure 79. Decomposition of Execute Create Client Diagram Command

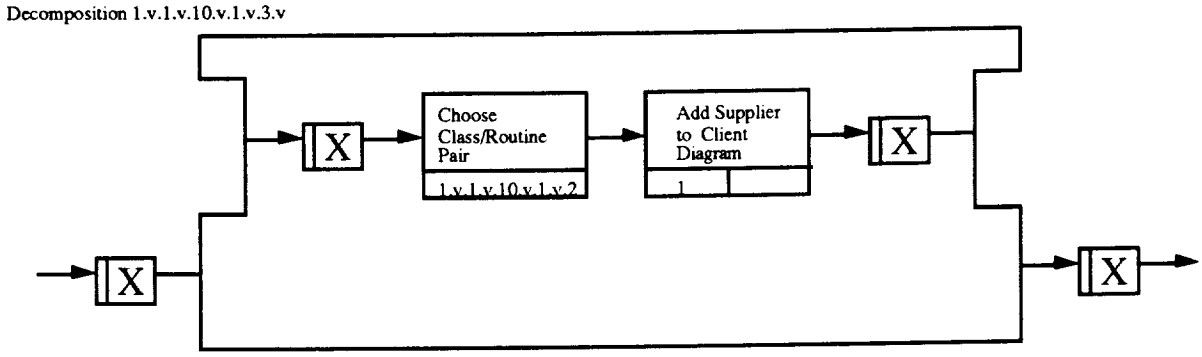


Figure 80. Decomposition of Add Suppliers to Client Diagram

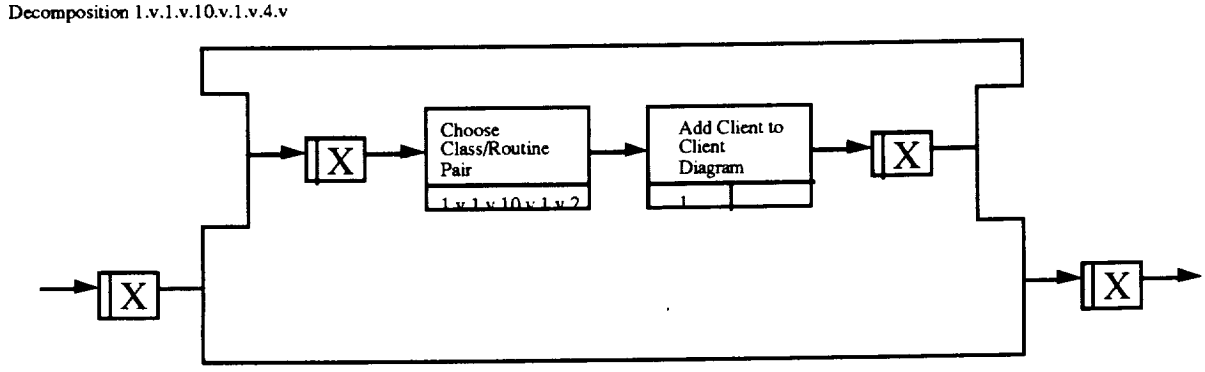


Figure 81. Decomposition of Add Clients to Client Diagram Command

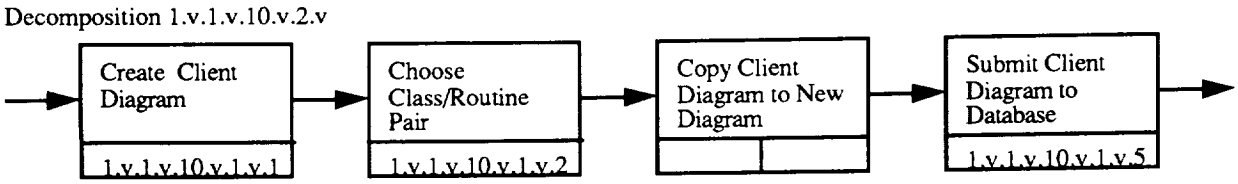


Figure 82. Decomposition of Execute Copy Client Diagram Command

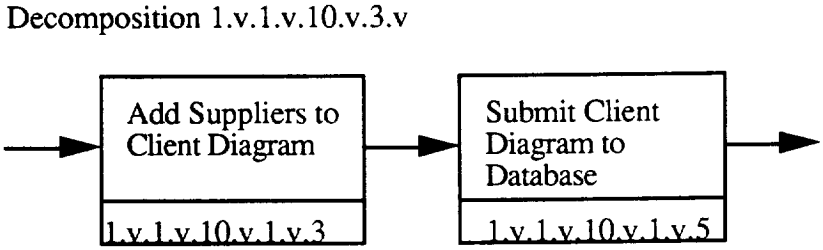


Figure 83. Decomposition of Execute Add Suppliers Command

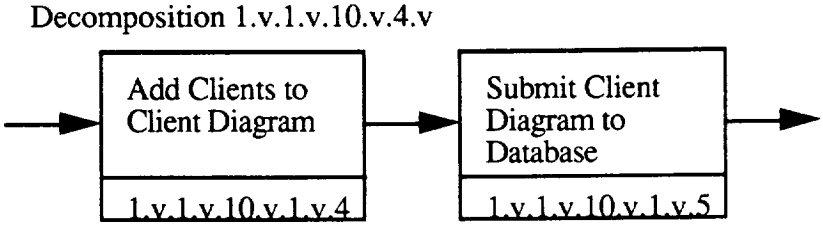


Figure 84. Decomposition of Execute Add Clients Command

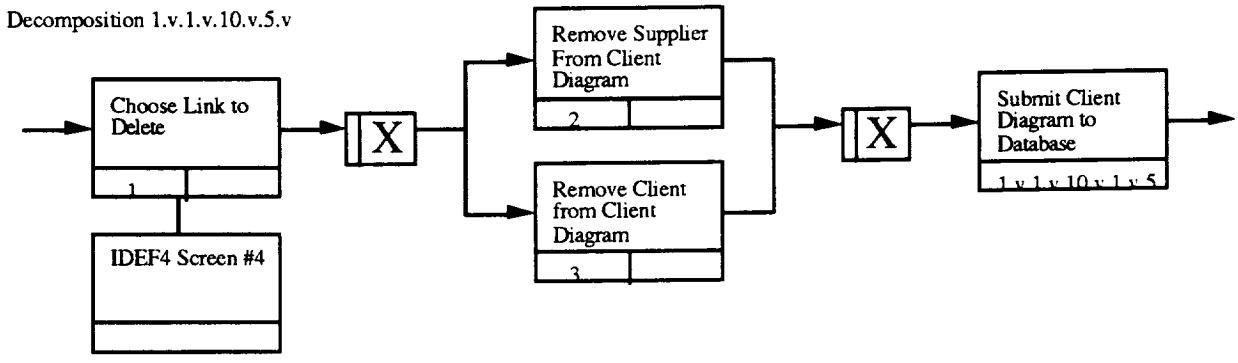


Figure 85. Decomposition of Execute Remove Subordinating Link Command

Decomposition 1.v.1.v.11.v

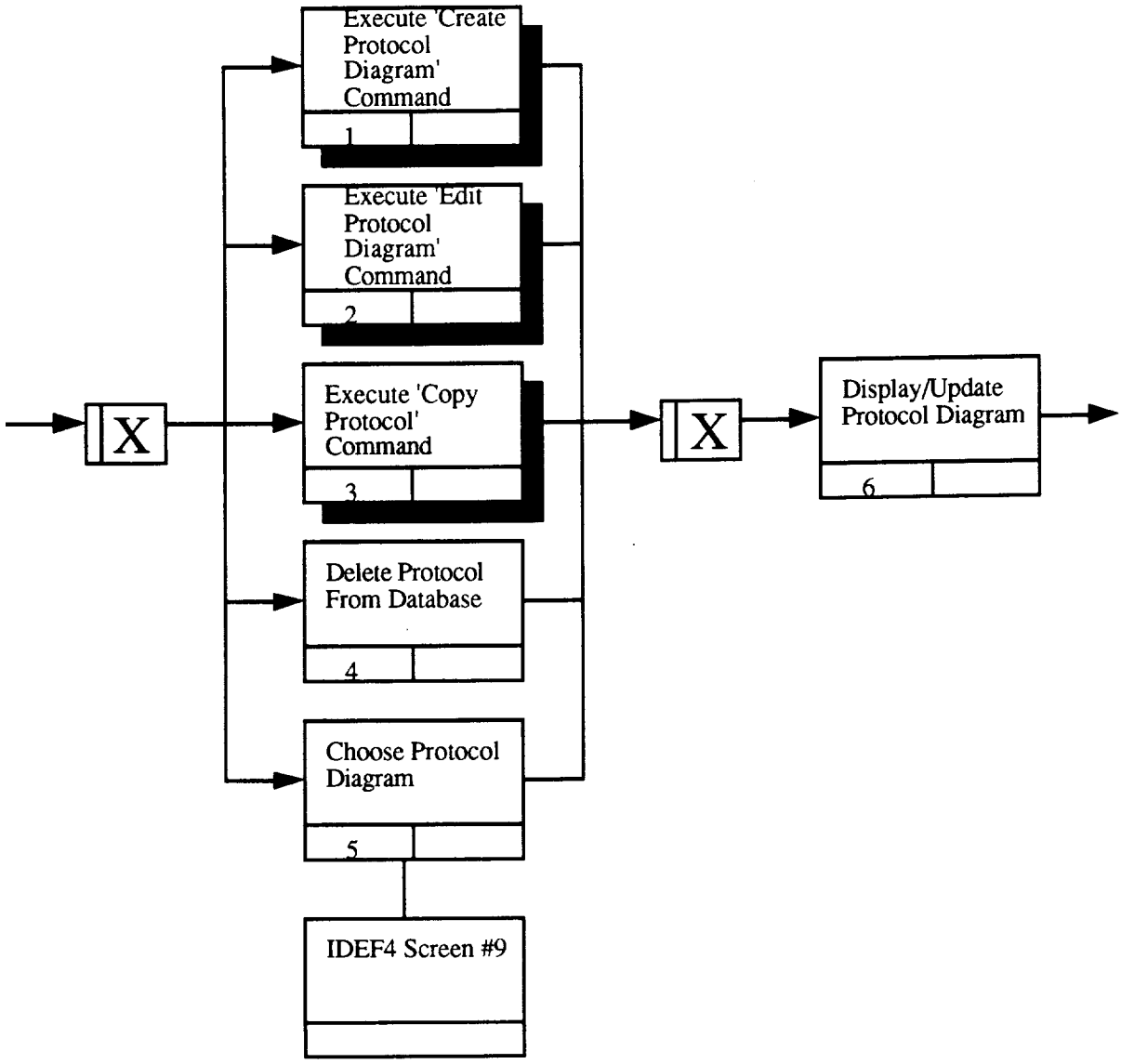


Figure 86. Decomposition of Execute Protocol Diagram Command

Figure 86 describes the operations defined for the manipulation of Protocol diagrams. Figures 87 through 89 provide additional details on these operations.

Constraints:

- When a Protocol diagram is deleted, all arguments associated with the feature's protocol are also deleted.

Decomposition 1.v.1.v.11.v

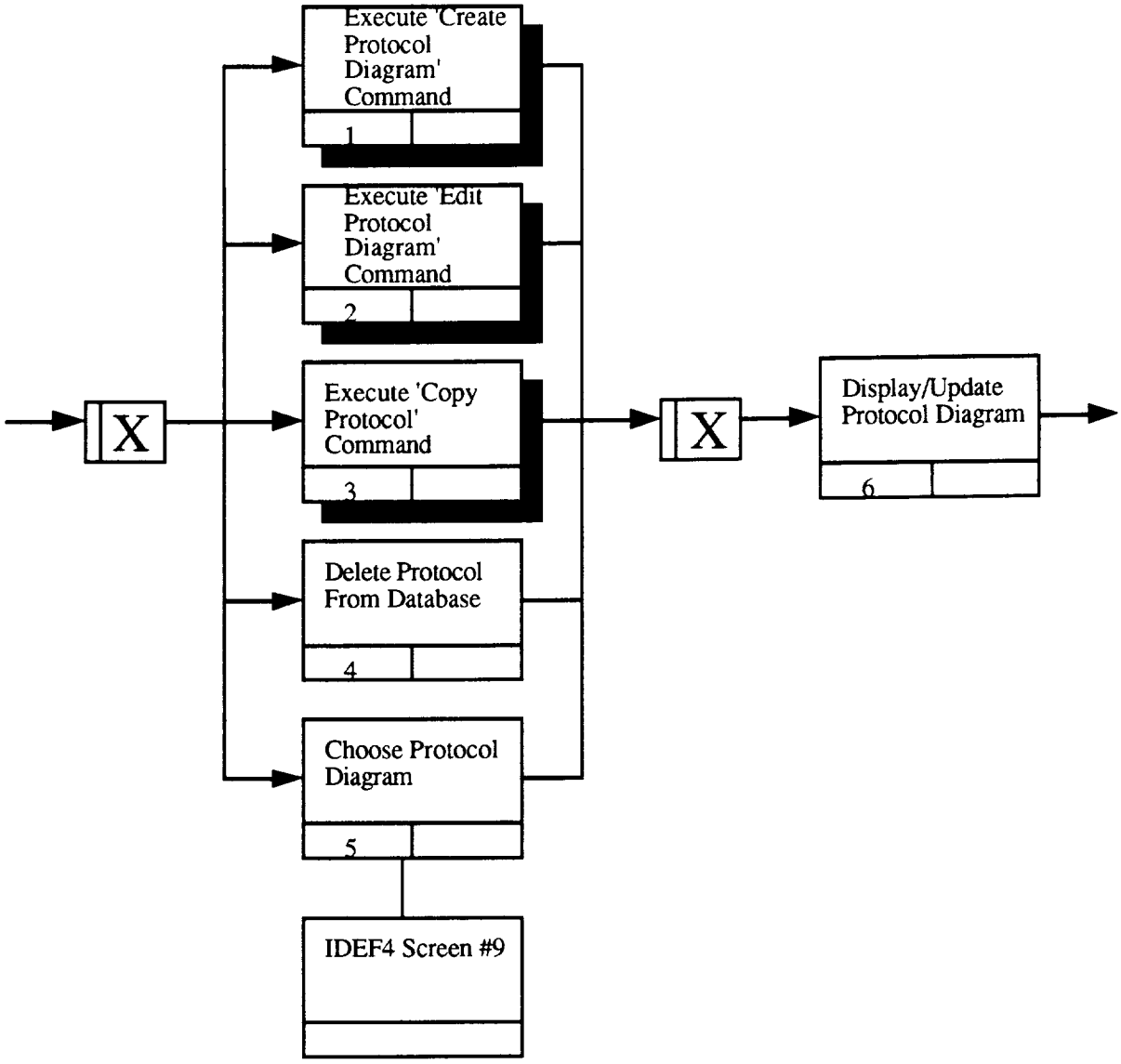


Figure 87. Decomposition of Execute Create Protocol Diagram Command

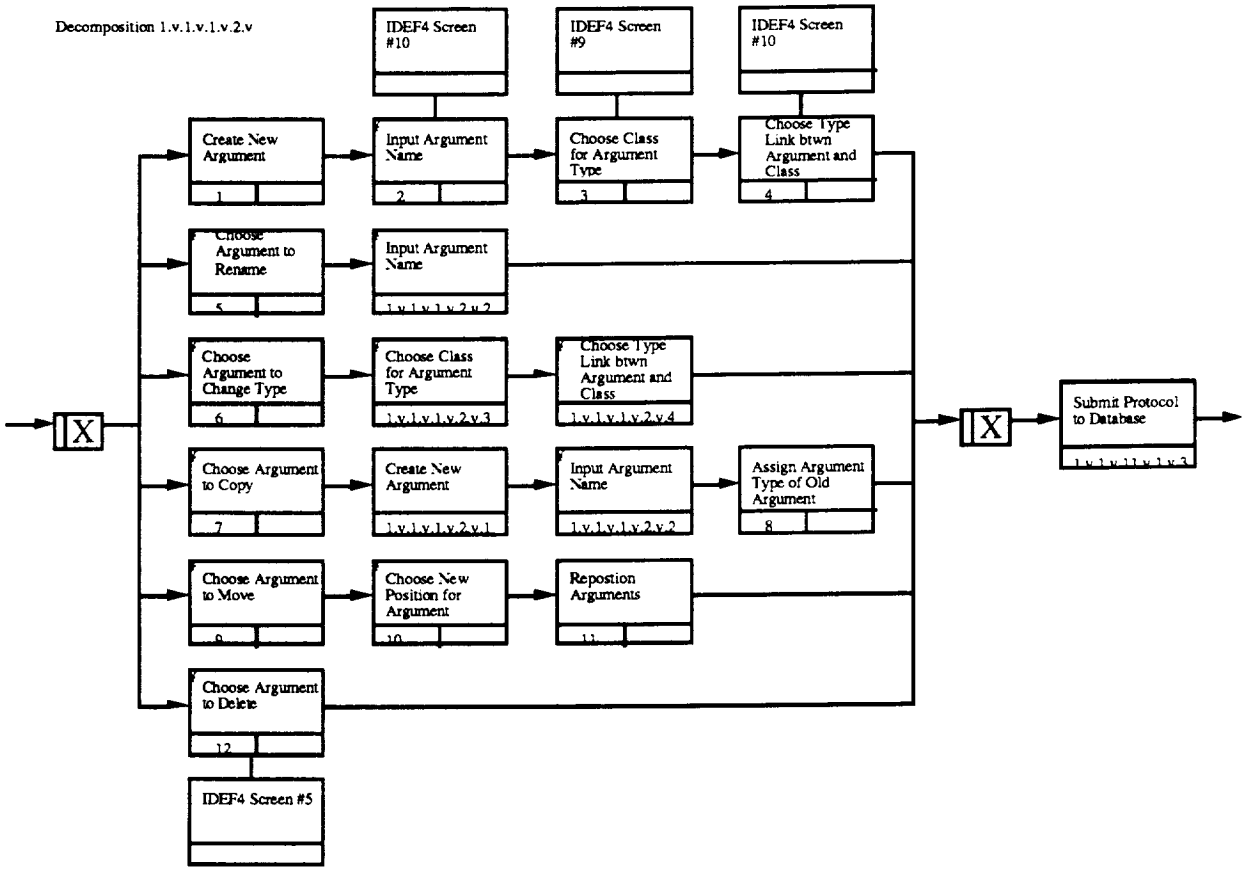


Figure 88. Decomposition of Execute Edit Protocol Diagram Command

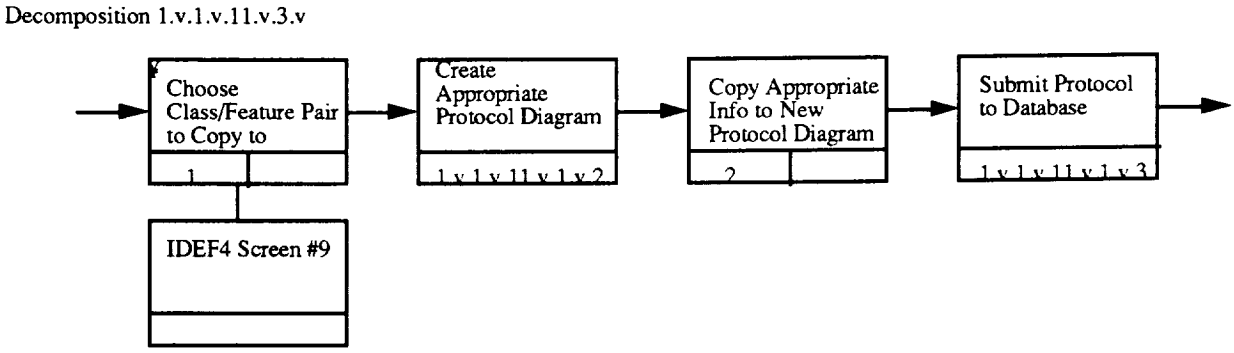


Figure 89. Decomposition of Execute Copy Protocol Command

Figures 90 through 100 display potential screens that will exist within the IDEF4 system and are the screens that are referenced within the process descriptions. These screens are meant only to give a rough idea of how the tool screens will appear. Figure 90

displays the Class Inheritance Diagram screen. Figure 91 displays the Type Diagram screen. Figure 92 displays the Method Taxonomy Diagram screen. Figure 93 displays the Client Diagram screen. Figure 94 displays the Protocol Diagram screen. Figure 95 displays the Class Browser screen. Figure 96 displays the Method Set Browser screen. Figure 97 displays the Feature Browser screen. Figure 98 displays the Pick From List Pop-up window. Figure 99 displays the Input/Edit Pop-up Window. Finally, Figure 100 displays the Generic Text Edit Window.

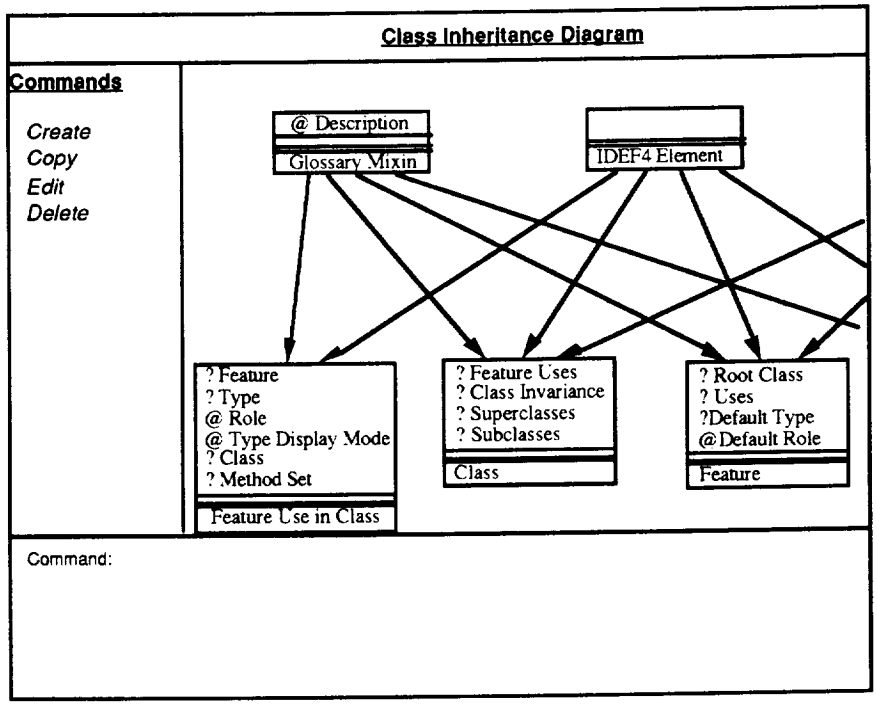


Figure 90. IDEF4 Screen 1 - Class Inheritance Diagram

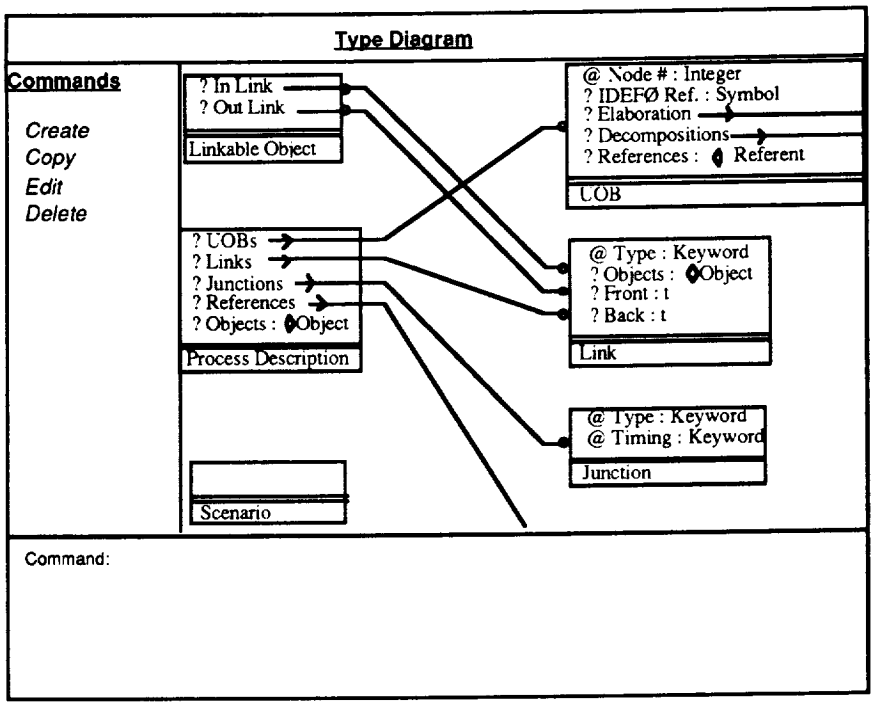


Figure 91. IDEF4 Screen 2 - Type Diagram

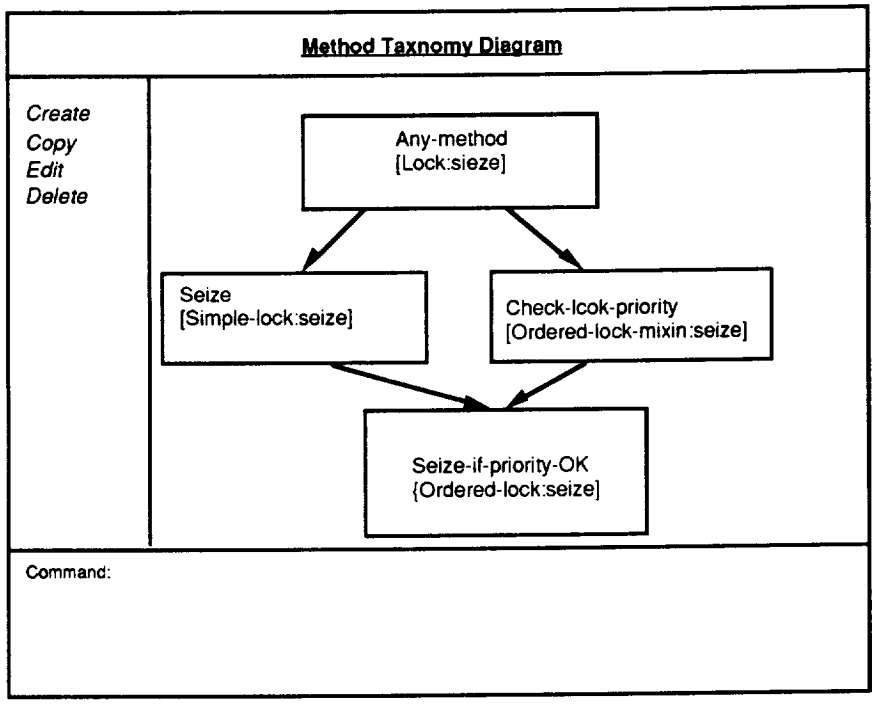


Figure 92. IDEF4 Screen 3 - Method Taxonomy Diagram Screen

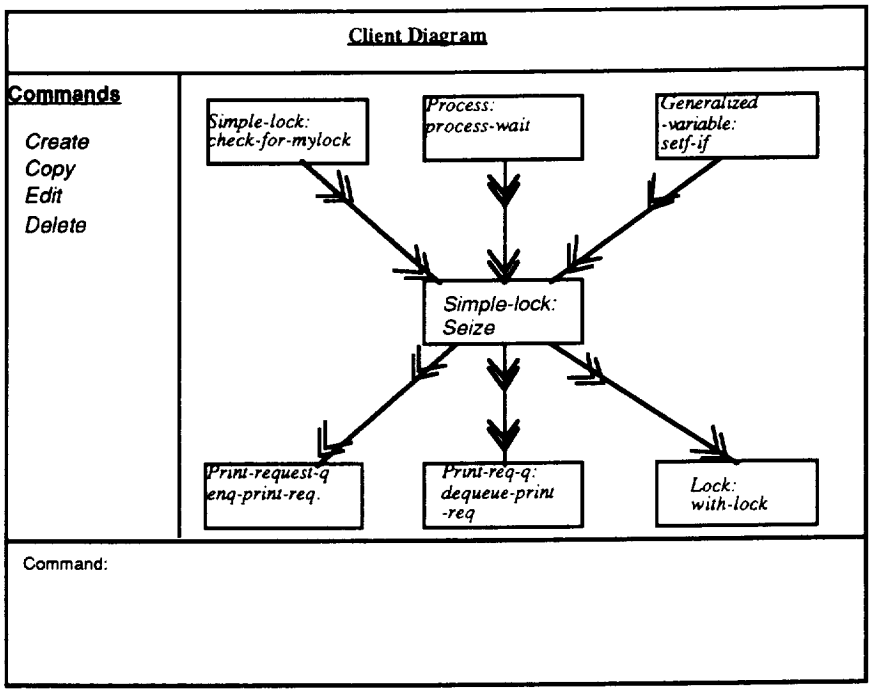


Figure 93. IDEF4 Screen 4 - Client Diagram

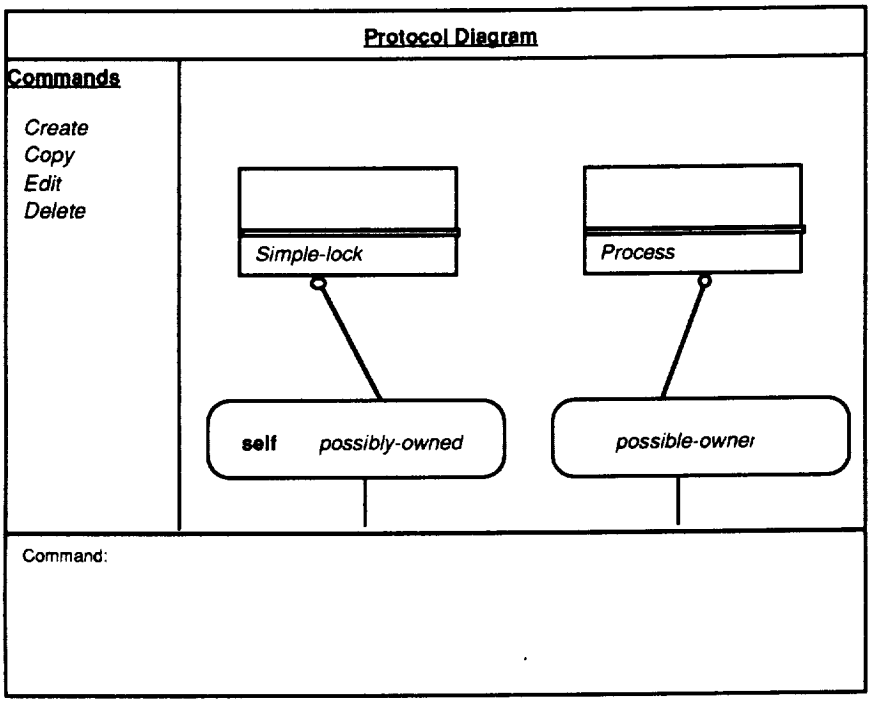


Figure 94. IDEF4 Screen 5 - Protocol Diagram

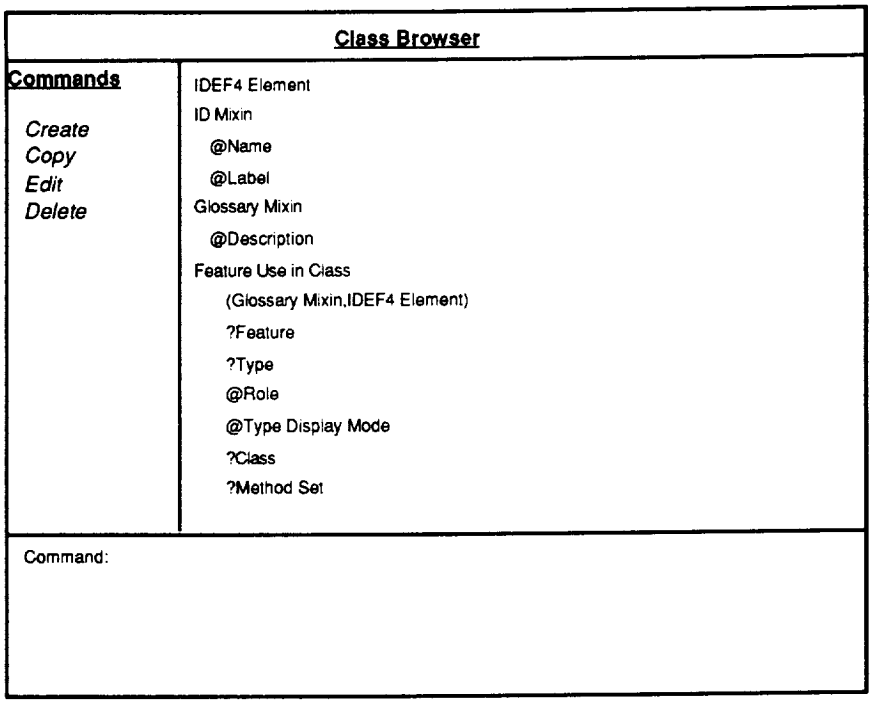


Figure 95. IDEF4 Screen 6 - Class Browser

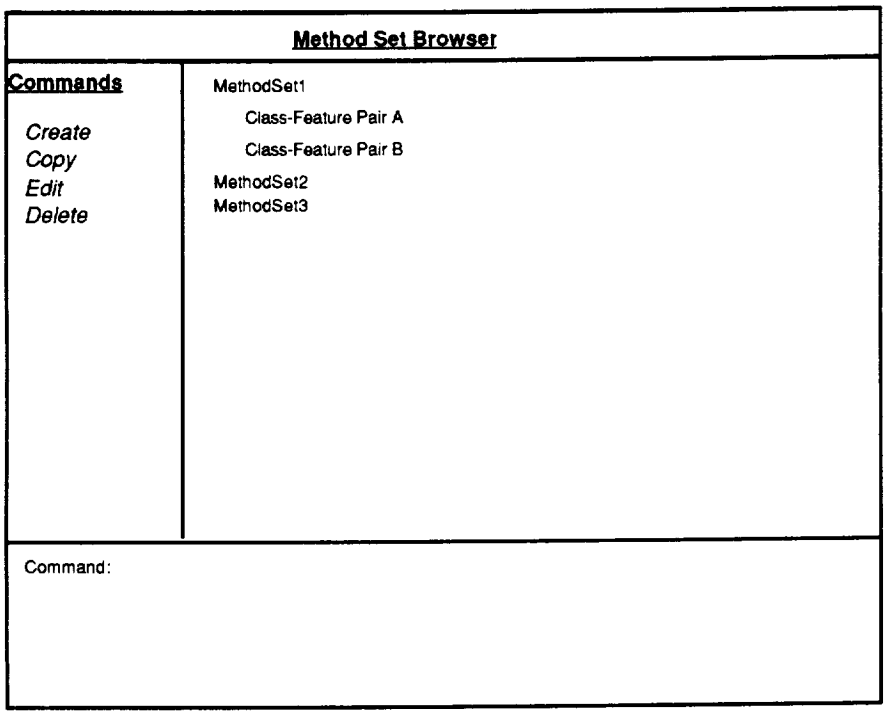


Figure 96. IDEF4 Screen 7 - Method Set Browser

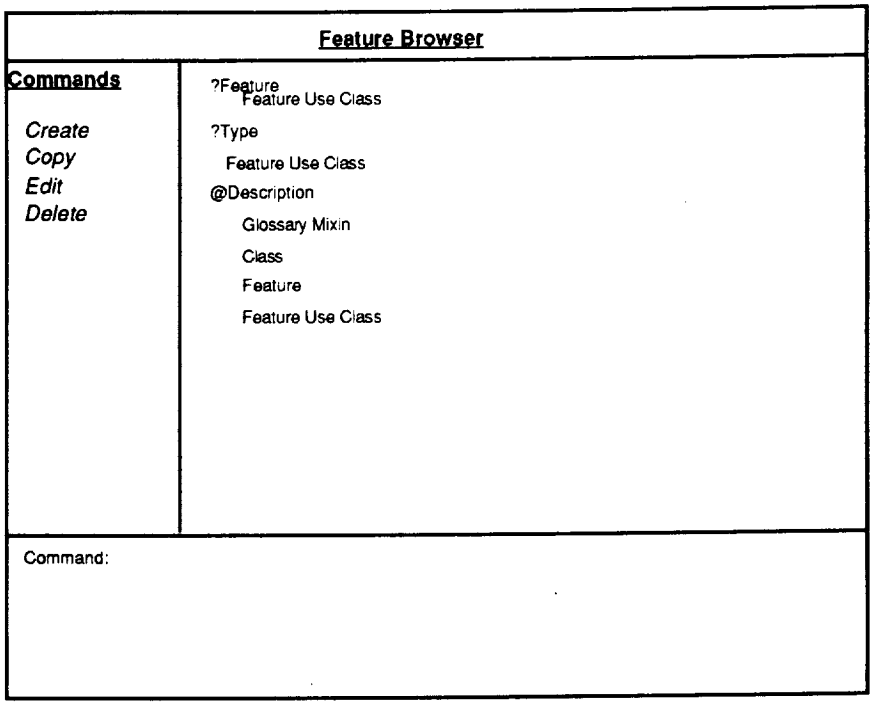


Figure 97. IDEF4 Screen 8 - Feature Browser

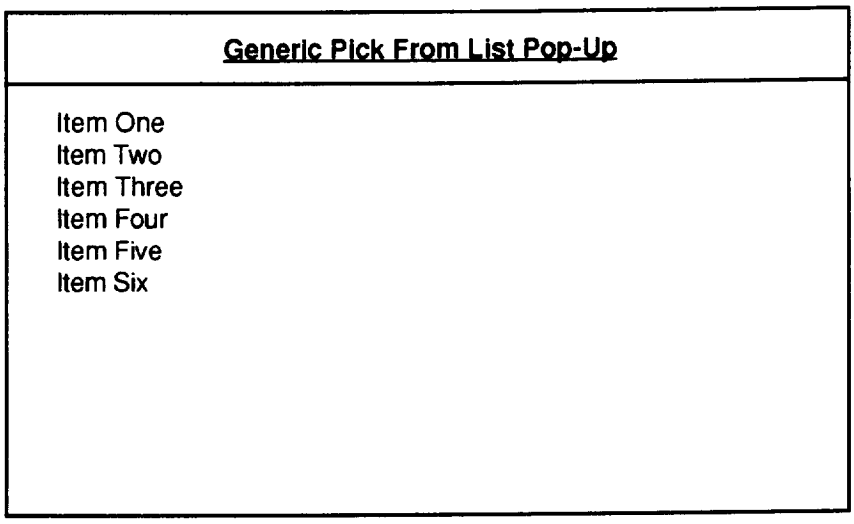


Figure 98. IDEF4 Screen 9 - Pick From List Pop-up Window

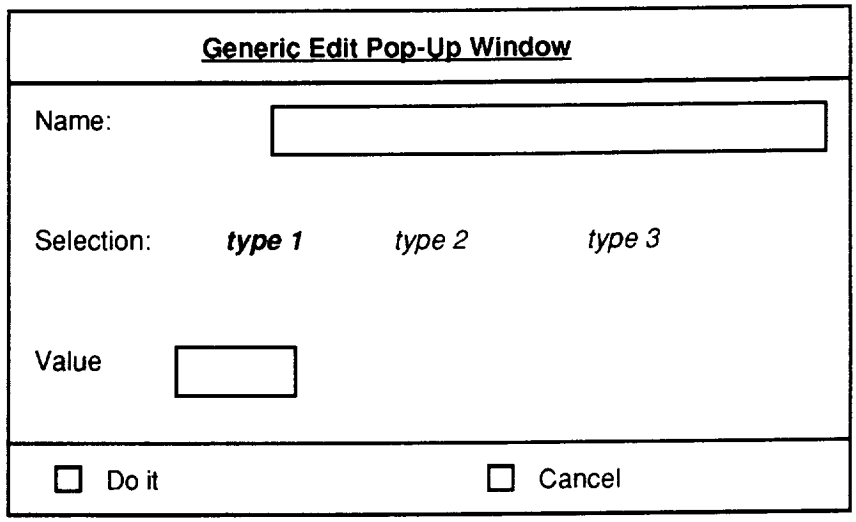


Figure 99. IDEF4 Screen 10 - Input/Edit Pop-up Window

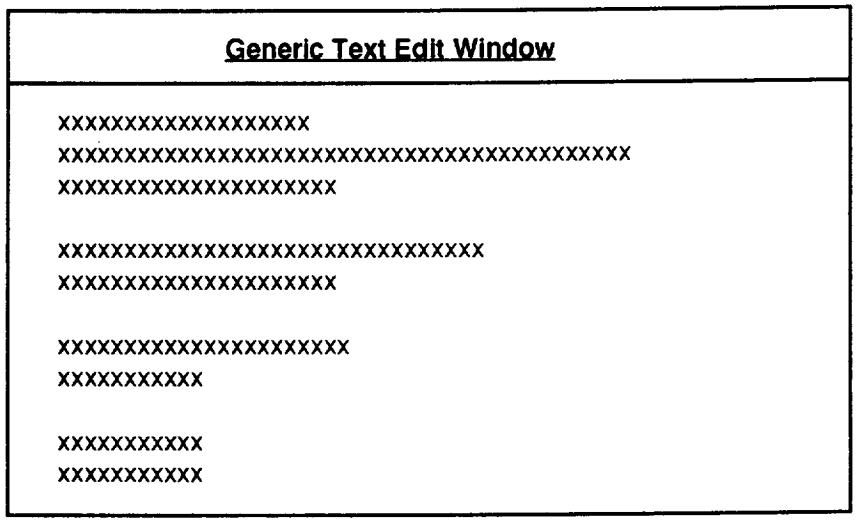


Figure 100. IDEF4 Screen 11 - Generic Text Edit Window

5.0 Conclusion

This document has presented the current designs for the automated IDEF3 and IDEF4 tools. It describes the philosophy behind the tool designs as well as the conceptual view of the interacting components of the two tools. Finally, it presents a detailed description of the existing designs for the tools using IDEF3 process descriptions and IDEF4 diagrams.

In the preparation of these designs, the IDEF3 and IDEF4 methodologies have been very effective in defining the structure and operation of the tools. The experience in designing systems in this fashion has been very valuable and will result in future systems being designed in this fashion. However, the number of IDEF3 and IDEF4 diagrams that were produced using a Macintosh for this document attest to the need for an automated tool to simplify this design process. An idea developed from the production of this document is to possibly tie the IDEF3 and IDEF4 tools to life cycle documentation generation systems. The result would be the production of documents similar to this one without the tedious work required to produce the diagrams.

Finally, as with any design, the designs of the IDEF3 and IDEF4 tools presented here are subject to modification during the implementation of the tool prototypes. However, it is hoped that the use of IDEF3 and IDEF4 in the design process has resulted in the development of stable designs that will require a minimal amount of revision during implementation.