RATM:
REQUIREMENTS ANALYZER AND TRACEABILITY MANAGER

by

David A. Long

Project report submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Systems Engineering

APPROVED:

W. J. Fabrycky

W. J. Fabrycky, Chairman

Ben S Blanchard

B. S. Blanchard

S. F. Midkiff

April, 1992

Blacksburg, Virginia

RATM:
REQUIREMENTS ANALYZER AND TRACEABILITY MANAGER

by

David A. Long

Committee Chairman: Wolter J. Fabrycky
Industrial and Systems Engineering

(ABSTRACT)

To fill the need for automated assistance during the system engineering process, a prototype computer-aided tool emphasizing requirements analysis and traceability management is designed and developed. An object-oriented approach is used which leads to the selection of Smalltalk as the implementation language. Peter Chen's Entity-Relationship-Attribute database structure is selected to maintain a complete and consistent system definition. "User-friendly" interfaces are implemented to allow efficient manipulation and representation of the system definition data. These interfaces include a system engineering editor, two engineering views (text and hierarchy), a requirement extractor, and a schema extender. The prototype is expandable and extendable to permit future upgrades to support additional views, reports, and dynamic verification of models. With system definition becoming increasingly important in designing error-free, cost-effective systems, this computer-aided system engineering tool provides an accessible source of automated assistance to support the engineer.

ACKNOWLEDGMENTS

I wish to thank Omnitech Systems, Inc. for supporting the development of the Requirements Analyzer and Traceability Manager. I am also very grateful to James Long for his constructive suggestions and comments after reading the first draft of this paper.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

As high technology systems continue to increase in complexity and required performance, preliminary system design has become critical. The engineer must begin by defining the problem, analyzing the needs, performing feasibility analyses, defining operational requirements, and creating a maintenance concept. The next phase is to perform functional analyses, create the physical architecture, and allocate functions onto this architecture. The goal of the system engineering process is to develop specifications and models that are consistent, explicit, testable, and traceable to the initial problem definition. Current system engineering tools either do not support this goal, are not available because they are proprietary, or are extremely expensive tools requiring specific workstations. Therefore, there is a need for an affordable tool to support engineers during the initial phases of system definition, analysis, and design. The objective of this project is to design and develop a prototype that can serve as the basis of an automated tool to satisfy this need.

1.1 The System Definition Process

System definition is a special subset of the complete system engineering process. Though a life-cycle perspective should be maintained throughout development, the definition process ends with the completion of design as shown in Figure 1.1. This process begins with a statement of the problem and then continues with the derivation of the system concept which includes feasibility, operational, and maintenance considerations. At the conclusion of this stage, the requirements specification consisting of explicit engineering requirements and constraints is produced. These requirements are then traced to the functional model which describes interactions and sequences of required processes. Once the system behavior has been determined through the process of functional analysis and the physical architecture has been specified, the component structure establishing which part will perform which process is defined. The requirements specification, behavioral model, and physical model derive from and, in turn, enhance the definition that is maintained in a central database to ensure consistency as presented in Figure 1.2.

The system definition, once determined, acts as the hub for all other design and development activities. The definition captures technical design requirements and

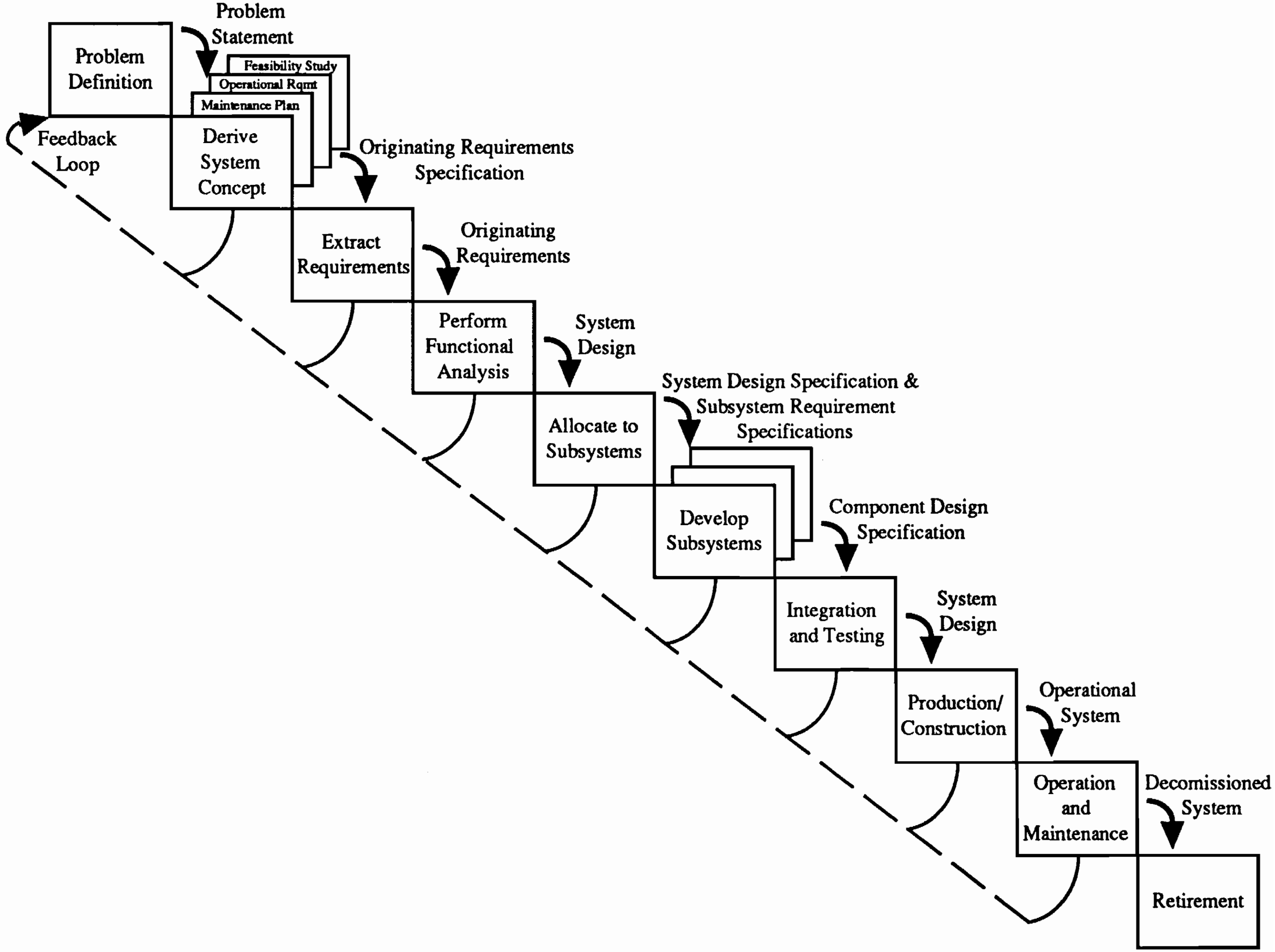


Figure 1.1 System Life Cycle

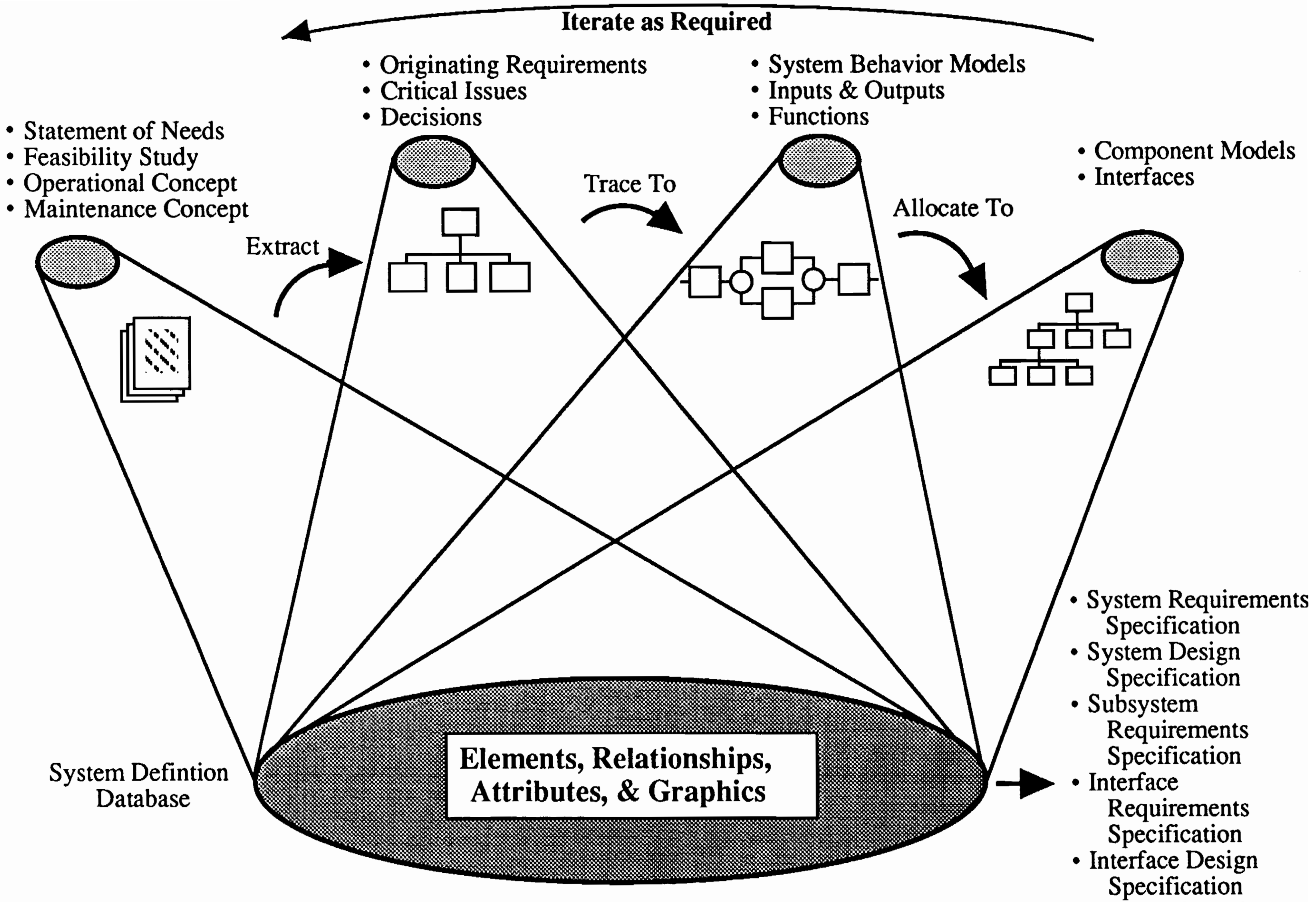


Figure 1.2 The System Definition Process

constraints (in the form of a requirements specification) required by engineers dealing with particular facets of the design instead of the often imprecise, qualitative requirements put forth in the initial system concept. Furthermore, the development and integration of components is guided by the system specification. For example, the specifications for the software components are established with consideration of hardware and reliability constraints as well as those imposed by the software. Moreover, the information flows both into and out of the central definition. Thus, the constraints of each discipline are taken into account during the initial design stage. Therefore, complete system definition is an essential step in the design process. (For a complete discussion of system engineering and the design process, see System Engineering and Analysis by Blanchard and Fabrycky [1].)

1.2 Current Status of System Engineering Tools

During system definition, an engineer must develop and track the following: requirements, function descriptions, data item descriptions, physical component descriptions, interface descriptions, graphs depicting behavior, graphs describing system architecture, analyses and trade studies, critical issues, design decisions, and printed reports. The interactions and inter-dependencies between these items are numerous and complex as shown in Figure 1.3. However, each item is often maintained separately which requires the engineer to manually update each affected part every time a change is made to any of the items. Often, the engineer fails to update one of the items due to the intricate set of interdependencies. This error leads to an inconsistent design. These inconsistencies, if not detected, will lead to flaws in the final design. Early detection (or avoidance) of errors is essential, because delayed error detection is extremely expensive. A rule of thumb used in industry is that the cost to fix an error increases by an order of magnitude for each phase of development completed. Errors not detected until the operation phase cost approximately ten thousand times as much to fix as the same error would have cost if detected during the design process. Even problems located during testing cost one hundred times as much to resolve as those found during design.

Three primary tools are used by engineers to facilitate the design process. First is pencil and paper. Though by far the most common and most flexible tool, this combination, when relied on exclusively, often leads to inconsistent design. Furthermore, the process is lengthy, and this approach is relatively slow. Second, many developers

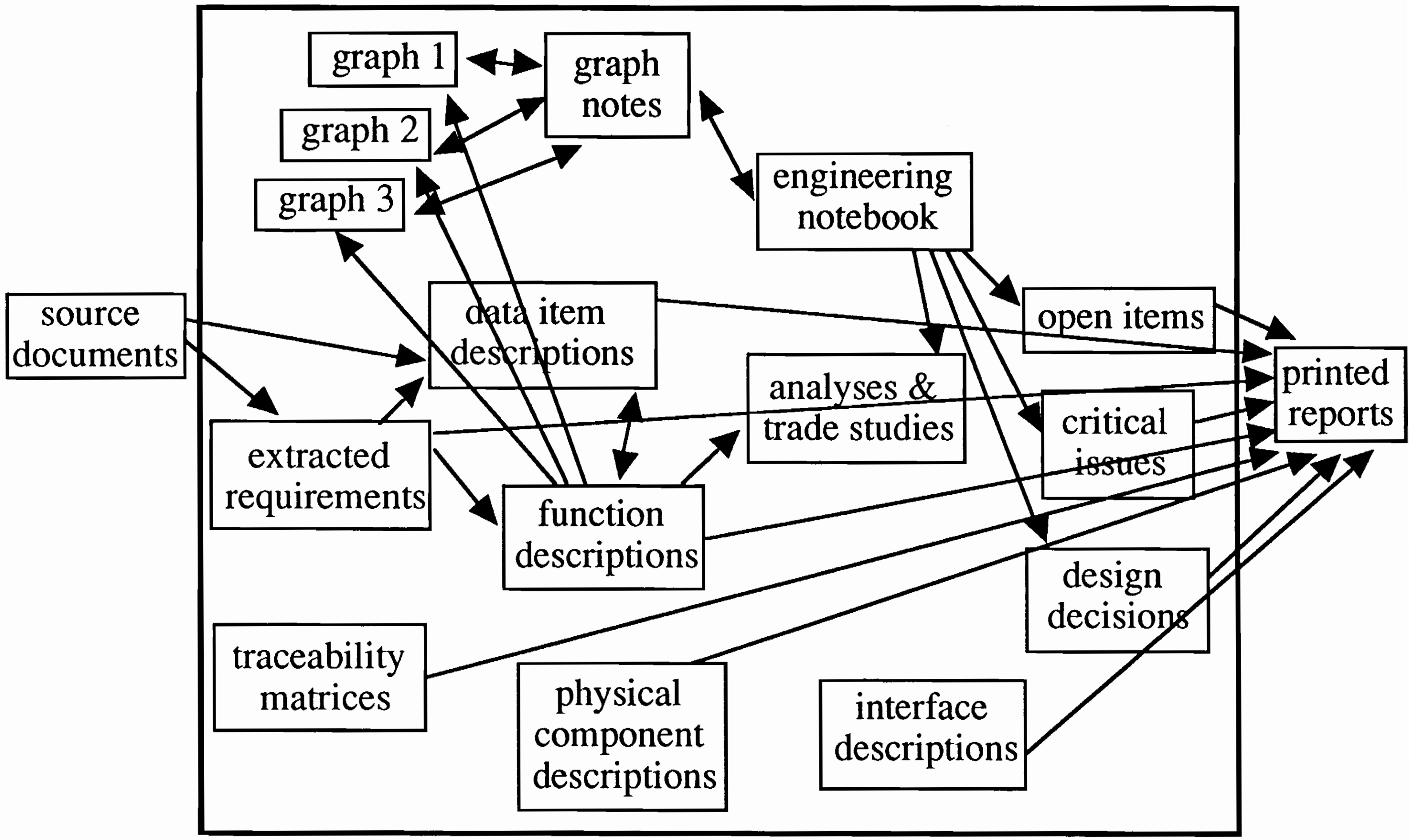


Figure 1.3 System Engineer's Desk Top

who employ automated methods misapply computer-aided software engineering (CASE) tools in an attempt to complete the design process. This is akin to substituting a screwdriver for a crowbar. Though the screwdriver can be used to pry up some items, it is neither as flexible nor as powerful as a crowbar. Third, a handful of computer-aided tools for improving the development of requirements and the system definition are available. Most of these, however, either have a hostile user interface, fail to integrate text and graphics to fully describe the system, are not available on a small platform such as the personal computer, or are excessively expensive. Therefore, most engineers do not have access to an automated tool that supports the system definition process.

1.3 Objective of this Project

The objective of this project is to design and develop an automated tool to support the system definition phase of the engineering life cycle with an emphasis on requirements analysis and traceability. Though the focus of the tool is on requirements and traceability, the entire system design as well as the design rationale is supported and captured. The process of updating related information when changes are made is automated in order to maintain a consistent database which eliminates a major source of error in design. The tool can be made available to the average engineer — neither its platform nor its cost will be prohibitive. Finally, the tool design facilitates future extension to support the other phases of system definition dealing with function flows and model verification.

The remainder of this report deals with specific issues and decisions relating to the design and implementation of the Requirements Analyzer and Traceability Manager (RATM). First, the class of existing system engineering tools are examined. Second, two key issues in the design of the tool are discussed: the language for implementation and the structure for the system database. Third, the RATM system which includes the system database and user interfaces is discussed. Fourth, areas for future work are outlined, because this tool is only the preliminary implementation of a complete system engineering tool. Conclusions drawn from the design and implementation of a system engineering tool are then stated. The RATM system environment, interfaces, and commands are then discussed in-depth in the appendices.

II. LITERATURE REVIEW

Out of the handful of computer-based systems to support the system definition process, there are two-hat stand above all others. The first system, initially implemented in 1976, is SREM. Though oriented towards software engineering, the tools included in SREM represent a breakthrough in requirements engineering. DCDS, a direct descendent of SREM, incorporates the developments made during an additional twelve years of research. While DCDS retains a bias towards computer systems, it includes key facilities that explicitly support the system engineer during the system definition process.

2.1 SREM: Software Requirements Engineering Methodology

As described by Bell, Bixler, and Dyer [2], the purpose of SREM is to use computers to reduce the number and severity of problems encountered during the requirements specification phase of software engineering. Developed in response to the enormous set of requirements for the Ballistic Missile Defense (BMD) system, SREM consists of three primary pieces:

- a set of techniques and procedures for software requirement decomposition,
- the Requirements Statement Language (RSL), an artificial language used to state requirements, and
- the Requirements Engineering Validation System (REVS), an integrated set of tools designed to support the specification of requirements in RSL.

Both RSL and REVS have a number of noteworthy features. First, RSL is a machine-processable language developed exclusively for the specification of software requirements. Furthermore, in anticipation of projects with special needs, the language is extendable at the concept level. In order to use the Requirements Statement Language, a translator must be used to move data between the user interface and the REVS centralized database. A set of tools is built around the database to handle the necessary functions for software requirements engineering. These tools are:

- a consistency checker to verify the static completeness of the database,
- interactive graphics through which the user could specify the flow paths for the software, and
- an automatic simulation generator to dynamically verify the software specified by the requirements.

The preliminary implementation of the REVS software was available in 1976 on a Texas Instruments Advanced Scientific Computer used at the Ballistic Missile Defense Advanced Research Center in Huntsville, Alabama. However, possibly more important than the REVS software itself, SREM represents the first step in a long line of key software and system engineering tools developed by TRW in response to the special demands of the BMD program.

2.2 DCDS: Distributed Computing Design System

Based upon twelve years of research and development, the Distributed Computing Design System is a direct descendent of SREM [3]. Developed by TRW as a "limited distribution" product requiring formal U.S. Army approval for access, DCDS supports the entire software life-cycle. Though DCDS consists of a number of integrated modules (which include SREM itself), the module of most interest to system engineers is the System Requirements Engineering Methodology (SYSREM).

SYSREM goes beyond the original software orientation of SREM to address the specific needs of system specification. It addresses and supports all of the major steps of system design. In particular, it includes a set of automated tools to support:

- functional decomposition including function flows, data flows, completion criteria, performance indices, and constraints,
- function allocation onto the physical model of the system,
- simulation of the behavior specified during functional decomposition, and
- automatic document generation.

This combination of tools built around a central database provides significant benefits in the areas of design traceability and impact analysis.

Though DCDS is a very useful system, it suffers from several significant problems. First, access to the system is limited because the software was developed for the U.S. Army. Second, an engineer must have access to a VAX computer with a Tektronix 4105 graphics terminal (or a Sun workstation for the latest rehosted version) in order to use the software. Finally, if a user is actually able to gain access to the software, he will find out that DCDS is "expert-friendly" which many users say is a euphemism for "user-hostile".

Like SREM, DCDS is the parent of another engineering tool — Requirements

Driven Design (RDD™) developed by Ascent Logic. RDD is a re-engineered, commercial derivative of the system-level portion of DCDS which has been implemented on the Sun and Apple Macintosh® II workstations. Because of its pricing structure and high platform demands (at least sixteen megabytes of RAM), RDD is only marginally more accessible to the average engineer than DCDS is.

RDD is a trademark of Ascent Logic Corporation.

Macintosh is a registered trademark of Apple Computer, Inc.

III. MATERIALS AND METHODS

Before development of the RATM system could begin, two major issues had to be resolved. First, a programming language had to be chosen in which to develop the tool. Second, the structure of the system database (the core of the tool) had to be defined. Because the language and structure chosen — Smalltalk and Entity-Relationship-Attribute, respectively — are not familiar to many people, a brief discussion of both is required.

3.1 Smalltalk as a Language for Development

Smalltalk is an object-oriented programming system developed at the Xerox Palo Alto Research Center during a time when many of the concepts for modern computer interfaces were being developed [4]. Recently, versions of the language have been released for personal computers which will provide many people with access to an object-oriented system.

Object-oriented languages differ from traditional languages in the approach taken to model situations. To develop a model in a traditional language such as Pascal or FORTRAN, the user models the steps required to proceed from beginning to end. Therefore, these languages are known as procedural languages. On the other hand, object-oriented languages such as Smalltalk focus on defining objects and the way they interact. In developing a model in an object-oriented language, the user must identify the objects being modeled, classify the objects according to similarities and differences, and define the methods by which objects interact with one another [5].

Because of the focus on objects instead of processes, the object-oriented approach has many advantages to that of procedural languages. First, object-oriented programming automatically enforces the concepts of encapsulation and information hiding. Because all information and related code are stored with an object, access to information within the object is restricted (avoiding unintended or unauthorized changes to data). Therefore, the objects can be considered to be black boxes that respond to external requests to store or return information. Second, because objects are black boxes, they support modularity which reduces interdependencies within a system, thereby minimizing the complexity [5]. Third, the user has the ability to access existing classes within the object-oriented programming system. With this ability, the user can extend and tailor the system to his specific needs. Last, the world is made up of objects which interact according to certain rules. Because an object-oriented language closely resembles the structure of the real world, it is easier to develop meaningful models using an object-oriented approach than using a process-oriented approach.

Beyond offering the advantages of this state-of-the-art approach on personal computers, Smalltalk also promotes "fearless programming" as described by Diederich and Milton [6]. "Fearless programming" encourages experimentation with algorithms, application code modules, and even the core system code. Smalltalk allows users to try out ideas and responds with immediate feedback. Furthermore, errors are simply messages that indicate that an object does not understand a message that it was sent. Because of this design, the user can proceed with little fear of causing irreversible damage [7]. Furthermore, Smalltalk supports a "divide and conquer" approach to software development which allows the user to code and test individual modules. Though Smalltalk does not excel at numerically intensive computations, its combination of an object-oriented approach with its support of "fearless programming" makes it ideal for the development of a system engineering database with graphic user interfaces.

3.2 The Entity-Relationship-Attribute Structure

In much the same way that Smalltalk is the natural aid for the development of the system definition tool, the Entity-Relationship-Attribute (ERA) structure proposed by Chen [8] is ideal for the representation of the data contained in a system database. The ERA structure is based on three primitive language concepts: entities, relationships, and attributes.

- Entities (also known as elements) correspond to nouns in English. Entities are the objects which the user wishes to define and serve as the basic units in the system database. All entities are classified into one of several classes (e.g., Component, Function) in the system database.
- Relationships are similar to verbs. To be precise, a relationship which defines a link between two entities corresponds to the mathematical definition of a binary relation. Relationships are not commutative, each relationship having a definite subject and object. However, for each relationship, there is a complementary relationship which defines the link from the object to the subject.
- Attributes further describe entities much like adjectives modify nouns. The attributes of an element serve to define critical properties of entities. For

instance, attributes of a component would include the component number and type.

Because the ERA structure consists of entities which are modified by attributes and related to other entities, it clearly corresponds to the object-oriented approach. Entities are represented as objects with the attributes stored as data within the objects. The relationships then define the interaction between objects.

To clarify the concept of the ERA structure, a simple model of the academic environment has been developed as an example. The entities are categorized into four classes — department, professor, student, and course — as shown in Figure 3.1. Each entity has basic attributes which define the entity. For example, as illustrated in Figure 3.1, each course has a description, a number, a set of prerequisites, and a list of semesters during which the course is offered. Furthermore, the entities are linked by relationships. As shown in Figure 3.1, a course is "taught by" a professor. The complementary relationship, though not shown in Figure 3.1, states that a professor "teaches" the course.

Not only does the ERA structure correspond closely to the problem it is trying to model, but it also serves as an artificial language in which the problem can be defined. Because the ERA structure is an artificial language, the precise meaning of each concept can be defined and documented [2]. Therefore, ambiguities present in other languages can be avoided. For example, requirements documents written in English are often very unclear. However, if the requirements were specified in an ERA structure, there could be no ambiguities, because the meaning of each entity class, relationship, and attribute would be clearly defined with a single meaning. Therefore, the language serves to clarify and enhance communications between system developers and users. Furthermore, the attributes and relationships for each entity class serve as a checklist of information which needs to be specified for an entity to be complete. Because the Entity-Relationship-Attribute structure not only naturally represents the required information but also serves as an artificial language which eliminates ambiguities and miscommunication, the best model for the system definition database is the ERA model.

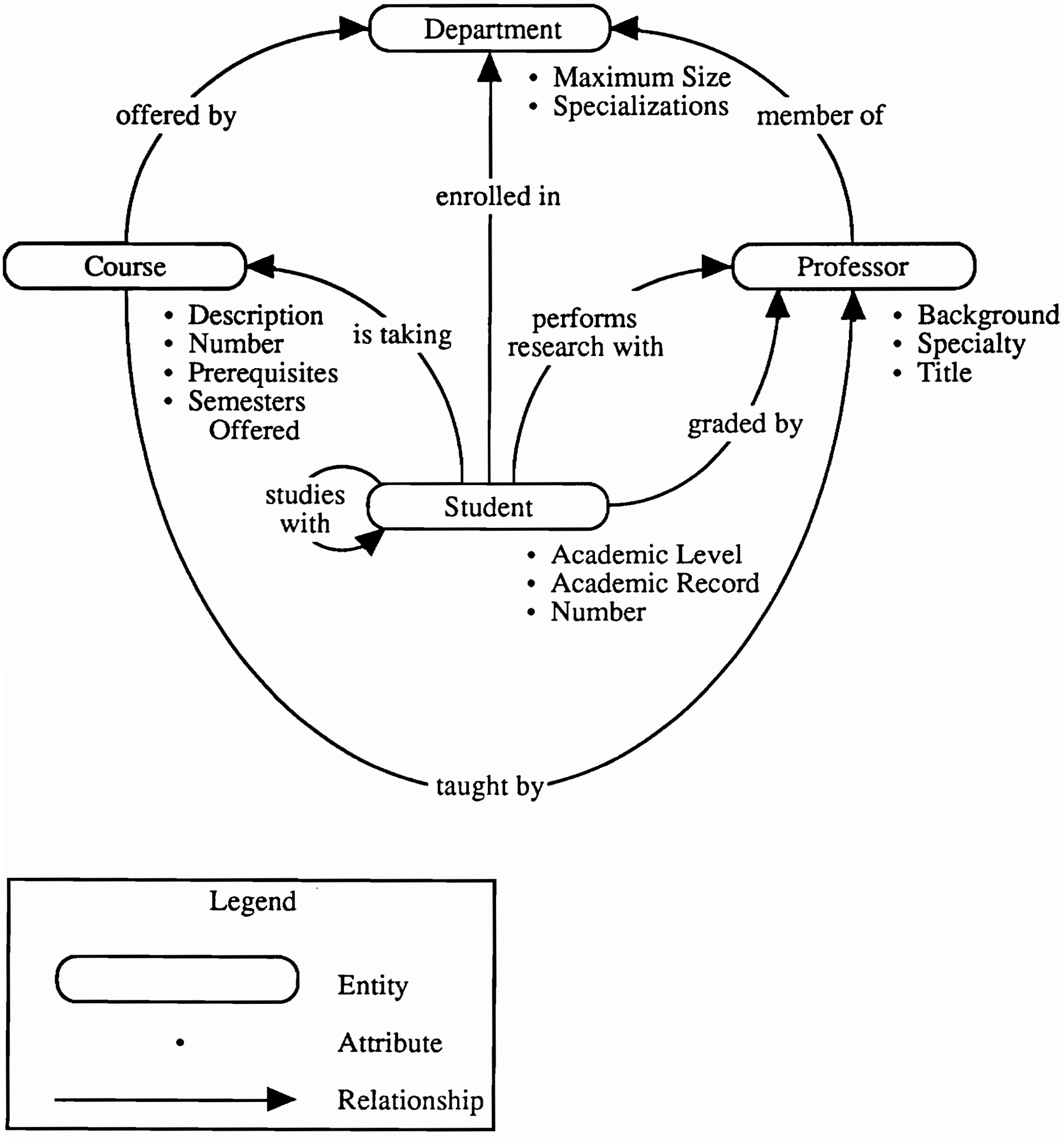


Figure 3.1 Simplified ERA Diagram for an Academic Environment

IV. RESULTS

Based on the ERA language and developed in Smalltalk, the Requirements Analyzer and Traceability Manager consists of five interfacing modules built around the system definition database. The heart of the tool is obviously this database which stores all the information generated during system development. The element editor, used to browse and manipulate the database, is the primary user interface of the tool. Two additional views, the text and hierarchy views, display a different subset of data in different formats to fulfill some of the needs of system engineers. Furthermore, because originating requirements are often specified and written by someone other than the engineer, the requirement extractor allows the user to transfer requirements directly from external documents to the system database. Finally, in case a user needs to add a new concept to the database structure, the schema extender allows the user to extend the baseline database structure. The remainder of this section consists of a brief summary of the database and the interfaces. (Appendix A contains a complete description of the RATM environment and interfaces as well as a complete command summary.)

4.1 System Definition Database

Built on the ERA structure, the structure of the system definition database (known as the schema) is designed to both maintain strong traceability throughout the system definition process and document the decisions made during design. The first priority in defining entities and relationships is to maintain complete traceability of all elements of the design. In this way, the origins and justification of all pieces of the design can be determined. Furthermore, the effects of changes in requirements can immediately be seen because all elements which descend from the changed requirement are linked to the requirement in the database. Moreover, the design rationale — the answer to the question "why", often kept only in the engineer's head — is captured in the database. Because engineers often move from project to project, this information is often lost and cannot be recovered if a design is reviewed at some point in the future. However, by allowing the user to document all engineering decisions, the database maintains a complete audit trail for future use. Finally, in order to avoid arbitrary design constraints, the schema is designed to allow the user to express the "what", "when", and "how well" but not the "how" of system design.

Though the RATM tool has modules that explicitly support only requirement analysis and design traceability, the database supports the entire system definition process. The entities, relationships, and attributes reflect the data present in the phases of design as first defined by Long, Dinsmore, Spadaro, and Alford [9] from 1968-1972. The Source class is an output of the problem definition and system concept derivation phases. OriginatingRequirement, Constraint, and PerformanceIndex all serve as design requirements. To complete functional analysis, the Function, Item, CompletionCriteria, and DomainSet classes are all needed. The allocation phase requires the System, ExternalSystem, Component, Interface, and ItemLink classes. The CriticalIssue and Decision classes are a repository for the design rationale. The remaining element type, Engineer, is simply a way to keep track of who is responsible for developing a particular part of the system. The complete set of entities, relationships, and attributes in the baseline schema are shown in Figure 4.1 and Figure 4.2. Appendix B also includes a complete set of Entity-Relationship diagrams for the baseline schema.

The baseline schema is defined to allow the user to work with requirements instead of forcing the user to define the database structure. However, some users will need to be able to extend the schema in order to fill the needs of their specific project. Therefore, the schema is extensible at the concept level. The user can add entity classes, relationships, and attributes to respond to specific needs or new requirements for defining systems. Between the baseline schema which supports the entire system definition process and the extendable nature of the schema, the system definition database can support the needs of any system development project.

4.2 Element Editor

The element editor, the primary interface for browsing and updating the system database, consists of nine different panes divided into two basic groups as shown in Figure 4.3. The five upper panes are list panes displaying the classes, elements, and relationships in the database. The lower four panes are labeled text panes displaying the name and three attributes of the selected element.

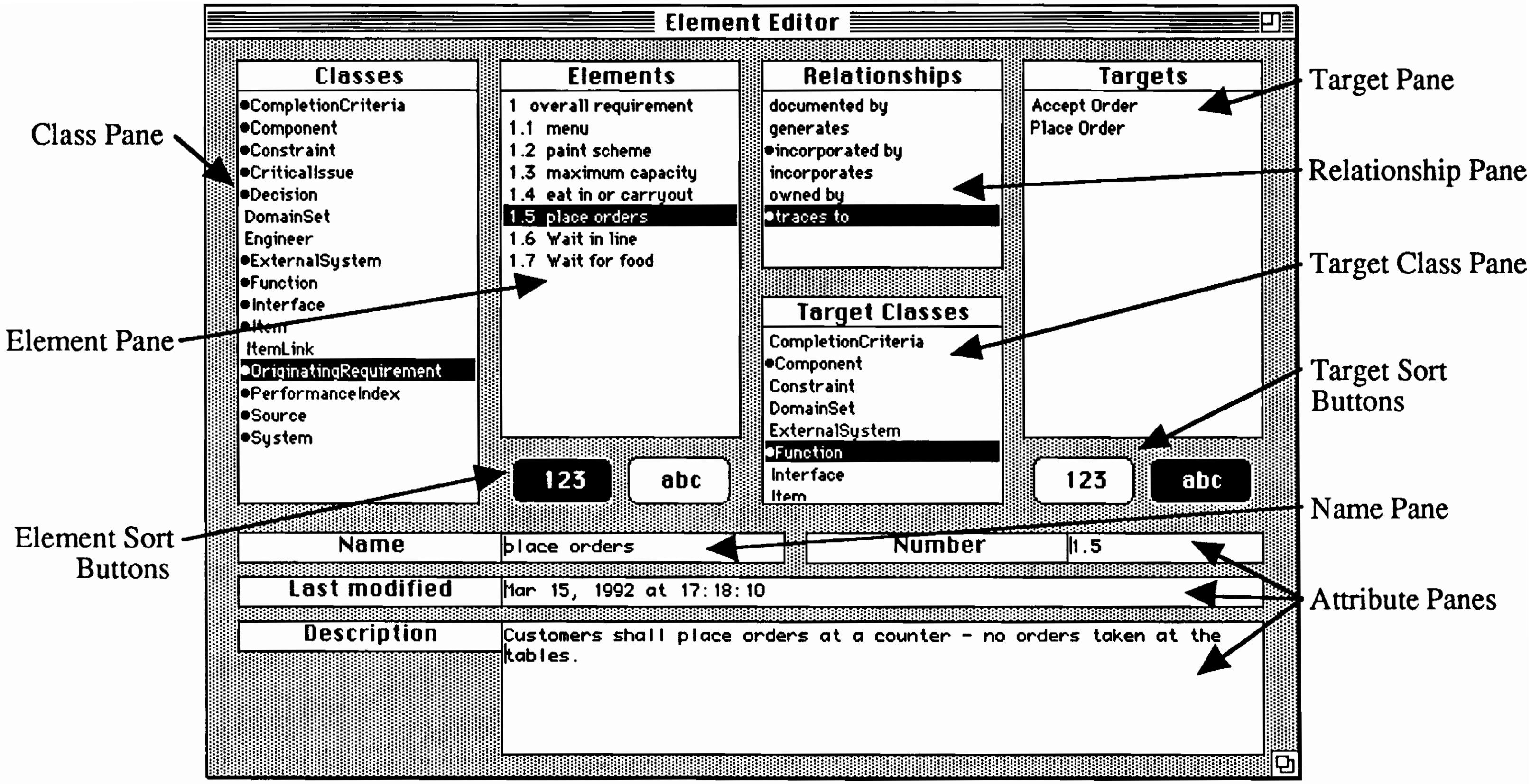
In the first list pane, the classes (e.g., Component, Function) are displayed. When the user selects the desired class, the elements within that class are shown in the element pane. The relationship pane lists all possible relationships for the selected class. When the user selects a relationship, all possible target classes are listed in the target class

Classes / Attributes	Abbreviation	Alternatives	Author	Category	Choice	Created on	Description	Index Value	Last modified	Number	Paragraph Number	Paragraph Title	Problem	Type	Units
CompletionCriteria	●		●			●	●		●	●				●	
Component	●		●			●	●		●	●				●	
Constraint	●		●			●	●		●	●					
CriticalIssue	●		●			●	●		●	●					
Decision	●	●	●		●	●	●		●	●			●		
DomainSet	●		●			●	●		●	●					
Engineer	●		●			●	●		●	●					
ExternalSystem	●		●			●	●		●	●				●	
Function	●		●			●	●		●	●					
Interface	●		●			●	●		●	●					
Item	●		●			●	●		●	●					
ItemLink	●		●			●	●		●	●					
Object	●		●			●	●		●	●					
OriginatingRequirement	●		●			●	●		●	●	●	●			
PerformanceIndex	●		●	●		●	●	●	●	●					●
Source	●		●			●	●		●	●				●	
System	●		●			●	●		●	●					

Figure 4.1 Baseline Schema Entity-Attribute Chart

Relationships	allocated to (performs)	"	"	built in (built from)	"	"	carried by (carries)	connected to (connects to)	constrained by (constrains)	decomposes (decomposed by)	"	documented by (documents)	exhibits (exhibited by)	exits by (exit for)	generates (is generated by)	incorporates (incorporated by)	input to (inputs)	is basis of (is based on)	is contained by (contains)	output from (outputs)	owned by (owns)	referenced by (references)	traced from (traces to)	"	"
Target Classes	Component	ExternalSystem	System	Component	ExternalSystem	System	ItemLink	Interface	Constraint	Function	Item	Source	PerformanceIndex	CompletionCriteria	CriticalIssue	OriginatingRequirement	Function	PerformanceIndex	Interface	Function	Engineer	DomainSet	CriticalIssue	Decision	OriginatingRequirement
Classes																									
CompletionCriteria									●			●			●						●			●	●
Component				●	●	●		●	●			●	●		●						●			●	●
Constraint									●			●			●						●			●	●
CriticalIssue															●						●				
Decision												●			●						●		●		
DomainSet									●			●			●						●			●	●
ExternalSystem								●	●			●			●						●			●	●
Function	●	●	●						●	●		●	●	●	●						●			●	●
Interface									●			●			●						●			●	●
Item	●	●	●				●		●		●	●			●		●			●	●	●		●	●
ItemLink									●			●			●				●		●			●	●
OriginatingRequirement												●			●	●					●				
PerformanceIndex									●			●			●			●			●			●	●
Source															●						●				
System								●	●			●	●		●						●			●	●

Figure 4.2 Baseline Schema Entity-Relationship Chart



Classes
●CompletionCriteria
●Component
●Constraint
●CriticalIssue
●Decision
DomainSet
Engineer
●ExternalSystem
●Function
●Interface
●Item
ItemLink
●OriginatingRequirement
●PerformanceIndex
●Source
●System

Elements
1 overall requirement
1.1 menu
1.2 paint scheme
1.3 maximum capacity
1.4 eat in or carryout
1.5 place orders
1.6 Wait in line
1.7 Wait for food

Relationships
documented by
generates
●incorporated by
incorporates
owned by
●traces to

Target Classes
CompletionCriteria
●Component
Constraint
DomainSet
ExternalSystem
●Function
Interface
Item

Targets
Accept Order
Place Order

Name: place orders
Number: 1.5
Last modified: Mar 15, 1992 at 17:18:10
Description: Customers shall place orders at a counter - no orders taken at the tables.

Figure 4.3 Element Editor

pane as illustrated in Figure 4.3. Finally, when the user selects a target class, any targets of that relationship are displayed. In this way, the user can browse the contents of the database and modify elements and relationships.

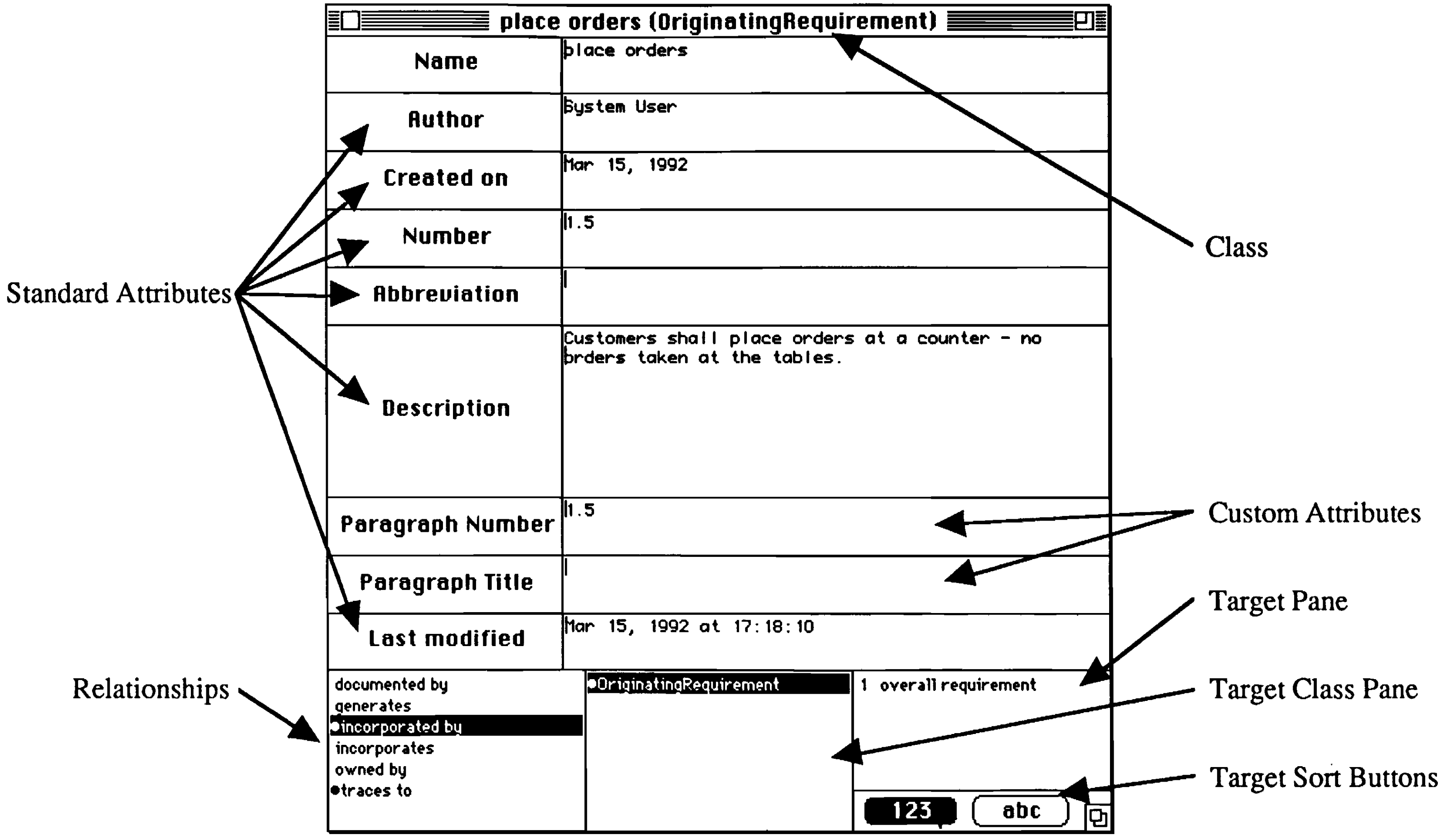
The selection of a specific element in the element pane causes the attributes corresponding to that element to be displayed in the lower text panes. The first pane displays the name of the selected element. The remaining three panes display attributes of the selected element. The attributes displayed in Figure 4.3 are "Number", "Last modified", and "Description". To change the attribute displayed in a specific pane, the user can click on the label for the pane. A menu containing the five standard attributes for every element ("Abbreviation", "Author", "Created on", "Last modified", and "Number") will pop-up, and the user can select the desired attribute. Only standard attributes can be displayed and edited on the element editor.

In order to maintain a consistent database, special care is taken in manipulating elements and relationships. When the user either adds a target to a relationship or removes a target from a relationship, the tool automatically updates the complementary relationship as appropriate. For example, if "Item1" is added to the "inputs" relationship of "Function1", "Function1" is automatically added to the "input to" relationship of "Item1". Furthermore, if an element is deleted from the database, it is removed from all relationships in the database. Because the attributes of an element are stored only with the element, no special care is required in handling the attributes.

Because a standard, "user-friendly" Apple Macintosh interface has been developed and implemented, very little time and effort is required for an engineer to learn to effectively use the element editor. Furthermore, the combination of automatically updated list and text panes facilitates browsing and manipulating the underlying database. All actions, ranging from the modification of an element attribute to the creation of a new element, are menu and mouse driven. Finally, the tool automatically takes the appropriate steps to maintain a consistent database. Therefore, the engineer can concentrate on system engineering instead of learning and using a tool.

4.3 Text View

Unlike the element editor, the text view is not designed for browsing the database. Instead, all of the information on a particular element in the system database is displayed in the text view, a structured format illustrated in Figure 4.4. The attributes are displayed



place orders (OriginatingRequirement)

Name	place orders
Author	System User
Created on	Mar 15, 1992
Number	1.5
Abbreviation	
Description	Customers shall place orders at a counter - no orders taken at the tables.
Paragraph Number	1.5
Paragraph Title	
Last modified	Mar 15, 1992 at 17:18:10

documented by
generates
●incorporated by
incorporates
owned by
●traces to

●OriginatingRequirement

1 overall requirement

Figure 4.4 Sample Text View

in the upper portion of the window in labeled text panes. The relationships, target classes, and targets that complete the element definition are displayed in the lower portion of the window as shown in Figure 4.4.

The first labeled field on the text view is the element name. The remaining labeled text panes (up to fourteen panes) display the attributes for the element. As shown in Figure 4.4 the attribute panes are displayed in the following order:

- Author (non-editable)
- Created on (non-editable)
- Number
- Abbreviation
- Custom attributes of the element in alphabetical order (up to 9)
- Last modified (non-editable)

The first four and the last attribute are all standard attributes for all elements in the system. All other element attributes are classified as custom attributes and are defined either exclusively for specific classes or are added to all classes by the user. Because of the nature of the custom attributes, they are not accessible from the element editor. Therefore, the only manner these attributes can be specified is through the text view. The sole exception to this is the case of OriginatingRequirements whose paragraph name and number can be specified when the element is created through the requirement extractor.

Three attributes ("Author", "Created on", and "Last modified") are non-editable attributes. Both the "Author" and "Created on" attributes are automatically determined at the time the element is created. In order to maintain complete design documentation this information must be maintained. The "Last modified" attribute is automatically set whenever a relationship or attribute is changed and assists in tracing modifications to the database.

Though the attribute panes in the text view are more extensive than those in the element editor, the list panes at the bottom of the text view are identical to those of the element editor. As in the case of the element editor, the user can select a relationship by clicking on the relationship name in the list pane. After a relationship is selected, the target classes for the selected relationship are shown in the target class pane. If there is only one target class for the selected relationship, it is automatically selected. Once a target class is selected, all targets in that class are displayed in the target pane.

4.4 Hierarchy Diagram

Though the element editor and text view both allow the user to modify the database, the hierarchy diagram can only be used to graphically display (but not manipulate) relationships between elements in the database. Each diagram is defined by two primary parameters: a top-level element and a set of relationships. All descendants of the top-level element, as determined by the set of relationships, are determined and displayed in a hierarchical format as shown in Figure 4.5.

The hierarchy diagram consists of two elements: icons and arcs. Each element on the chart is represented by an icon (also called a node). As illustrated in Figure 4.5, the icon contains the name, number, and class of the element (though the user can toggle the display of the element class off if desired). Furthermore, if the icon appears in multiple places on the diagram, a small black square is drawn in the upper-right hand corner of the icon as shown in Figure 4.5. The arcs represents relationships between objects and targets. If the set of relationships for a hierarchy contains more than one relationship, each arc on the diagram is labeled to indicate which relationship links the two elements as shown in Figure 4.5. However, when only a single relationship is used to define a diagram (as in the case of a Function Hierarchy), relationship labels are not displayed in order to keep the clutter on the diagram to a minimum.

Hierarchy diagrams play a critical role in traceability management. By defining the appropriate set of relationships, all elements which trace to a specific element can be determined and displayed in an clear, graphic format. For example, if after six months of system development, a change order revising an original requirement is received, the engineer can quickly determine all affected elements. Therefore, the hierarchy diagram is a critical piece of the RATM system.

4.5 Requirement Extractor

Unlike the element editor, text view, and hierarchy diagram which can be used to display the contents of the system database, the requirement extractor can only be used to add information to the system database. If the user has a document (in electronic format) containing the originating requirements for the system under consideration, the requirement extractor can be used to accelerate the process of entering the requirements into the system.

The requirement extractor window consists of ten panes and numerous buttons as

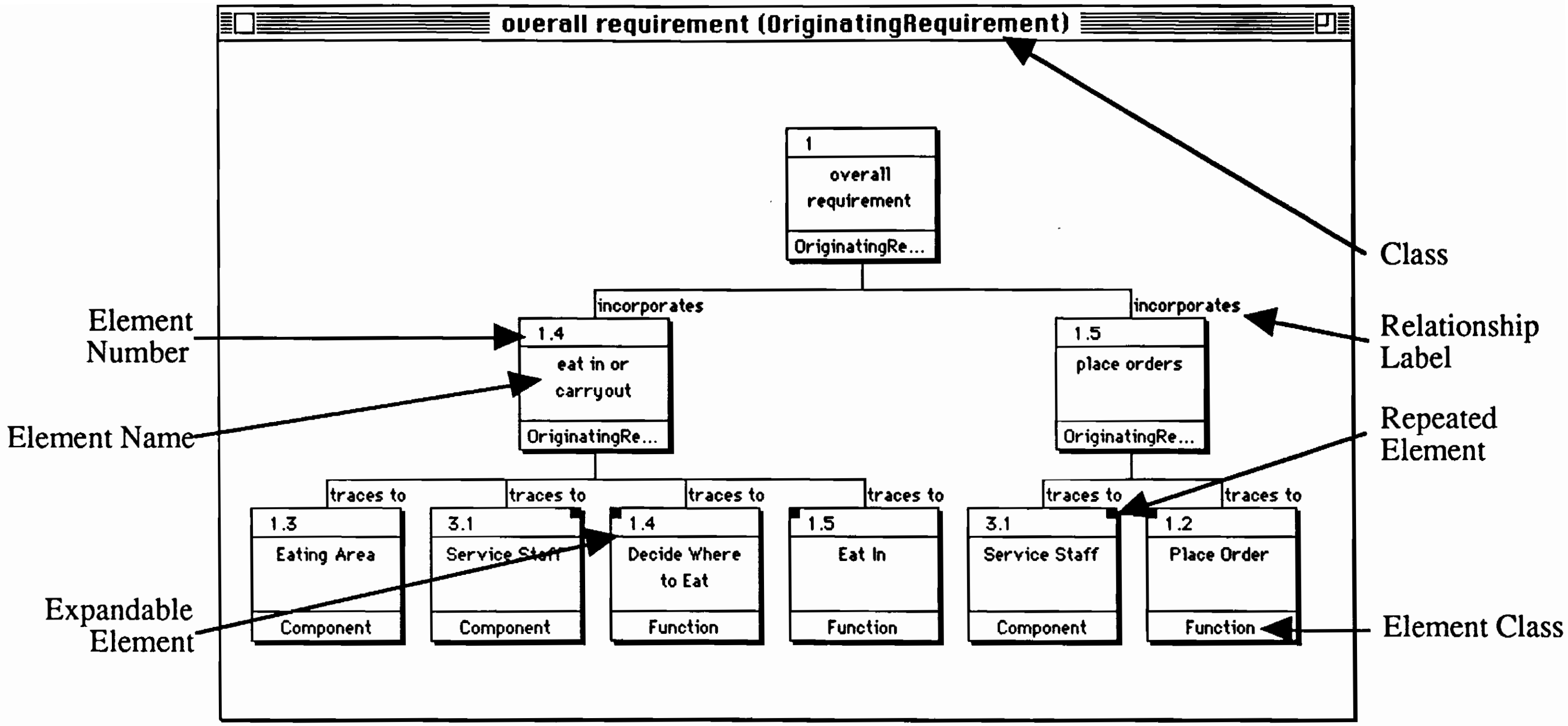


Figure 4.5 Sample Traceability Hierarchy

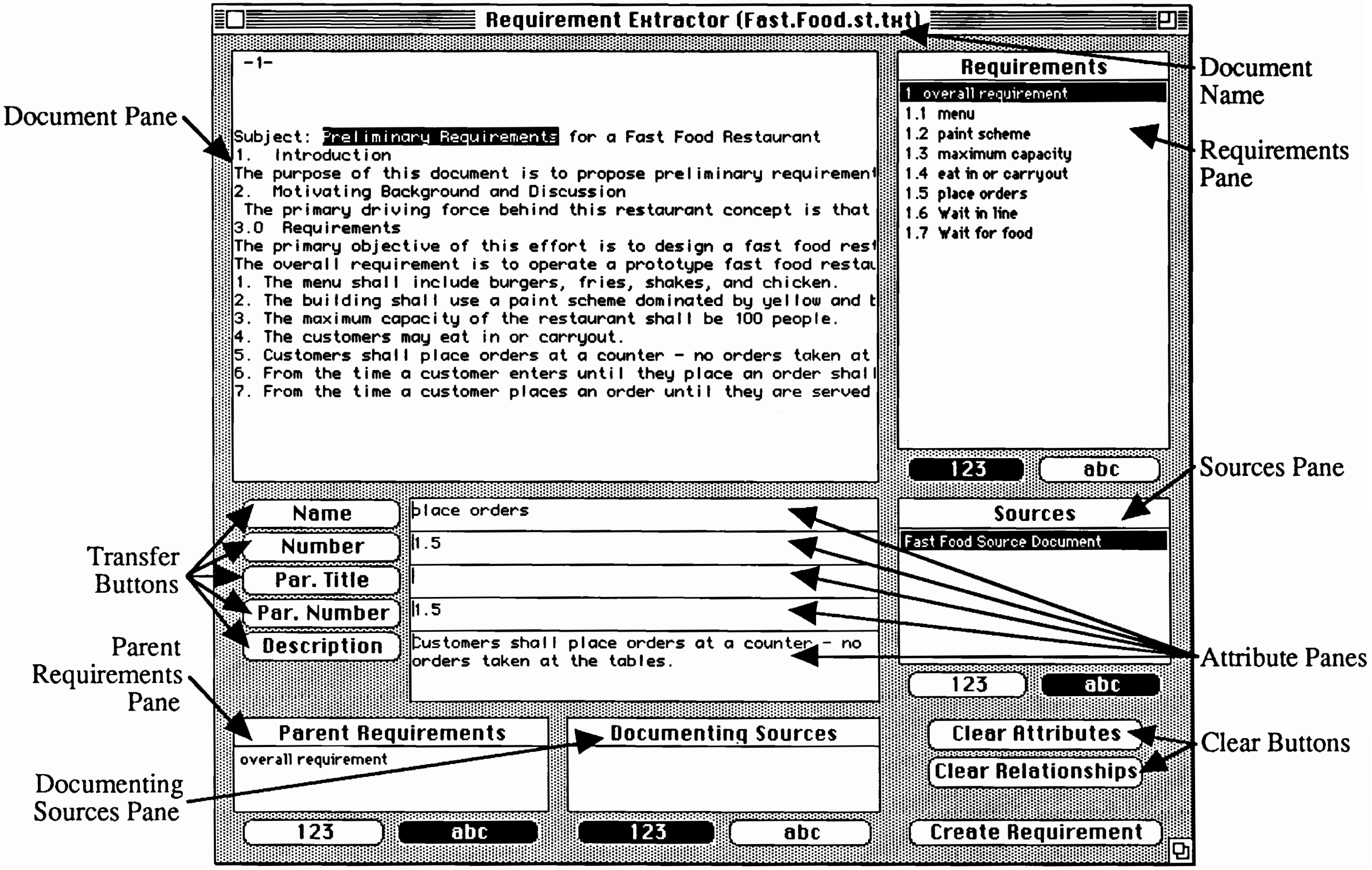
shown in Figure 4.6. First, the text pane in the upper-left hand portion of the window is the document pane. The five text panes directly below the document pane are the attribute panes. Below the attribute panes are the two list panes that display the targets for the relationships that can be specified in this window. Along the right side of the window are the window are two list panes which display all elements in the OriginatingRequirement and Source classes.

When the user opens the requirement extractor on a specified file, the contents of the file are displayed in the document pane. Because this is a text pane, the user can highlight text or perform any other standard editing function. Most often, the user will simply want to extract segments of text from the document and enter it into the database. Therefore, transfer buttons have been implemented to facilitate this process. To move text from the document into the definition of a requirement being created, the user can simply highlight the desired text in the document window and press the button corresponding the desired attribute. For example, the highlighted text in the document pane in Figure 4.6 can be transferred to the paragraph name pane by pressing the button labeled "Paragraph Name".

In addition to being able to specify the values of attributes when the requirement is being created, the user can add targets to two relationships. The source which documents the requirement can be specified in the documenting sources pane. The requirements which incorporate the requirement being created can be specified in the parent requirements pane. The list panes along the right side of the window are selection panes which allow the user to add targets to these two relationships.

4.6 Schema Extender

While the element editor, text view, hierarchy diagram, and requirement extractor are interfaces between the user and the contents of the system database, the schema extender allows the user to modify and extend the ERA language that defines the structure of the system database. The schema extender consists of four list panes as shown in Figure 4.7. In the first list pane, the classes (e.g., Component, Function) are displayed. When the user selects the desired class, the relationships and attributes for the class are displayed. The relationship pane lists all possible relationships for the selected class, and the attribute pane lists all attributes (both those that are inherited and those defined for the selected class). When the user selects a relationship, all possible target



Requirement Extractor (Fast.Food.st.txt)

-1-

Subject: Preliminary Requirements for a Fast Food Restaurant
1. Introduction
The purpose of this document is to propose preliminary requirement
2. Motivating Background and Discussion
The primary driving force behind this restaurant concept is that
3.0 Requirements
The primary objective of this effort is to design a fast food rest
The overall requirement is to operate a prototype fast food restau
1. The menu shall include burgers, fries, shakes, and chicken.
2. The building shall use a paint scheme dominated by yellow and t
3. The maximum capacity of the restaurant shall be 100 people.
4. The customers may eat in or carryout.
5. Customers shall place orders at a counter - no orders taken at
6. From the time a customer enters until they place an order shal
7. From the time a customer places an order until they are served

Requirements
1 overall requirement
1.1 menu
1.2 paint scheme
1.3 maximum capacity
1.4 eat in or carryout
1.5 place orders
1.6 Wait in line
1.7 Wait for food

Name: place orders
Number: 1.5
Par. Title:
Par. Number: 1.5
Description: Customers shall place orders at a counter - no orders taken at the tables.

Sources
Fast Food Source Document

Parent Requirements
overall requirement

Documenting Sources

Figure 4.6 Requirement Extractor

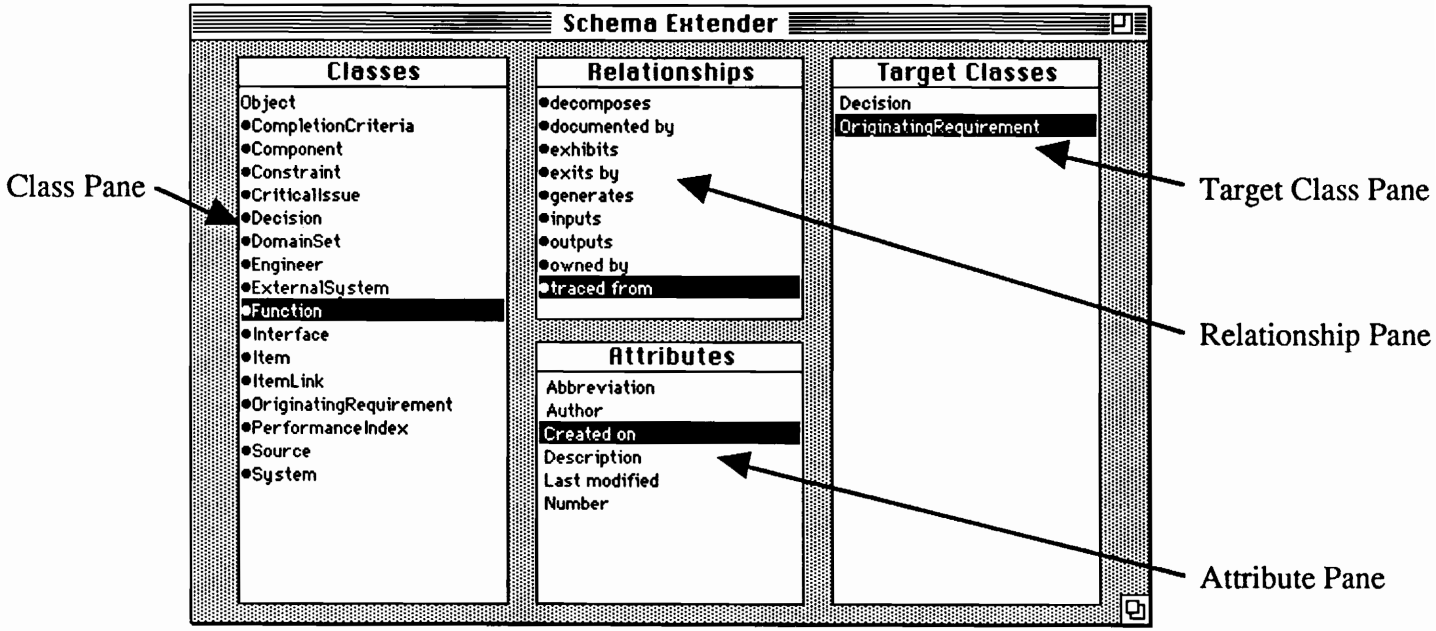


Figure 4.7 Schema Extender

classes are listed in the target classes pane. In Figure 4.7, the relationship "traced from" is selected, and the possible target classes (Decision and OriginatingRequirement) are displayed.

As shown in Figure 4.7, the first class in the class list pane is the superclass Object. This class is an abstract class. Unlike the other classes in the system, no elements of type Object can be defined. The sole purpose for this class is to define the structure of the other classes which inherit the characteristics of the superclass. Therefore, the attributes of class Object are automatically inherited by all other classes. However, though classes inherit attributes, they do not inherit relationships. Therefore, relationships cannot be defined for the superclass.

In order to maintain a meaningful database structure, the schema extender performs a basic set of checks to determine if the schema is consistent. A relationship is consistent if it has at least one target class. A class is consist if it has at least one relationship and if all relationships defined for the class are consistent. Obviously, if a relationship has no target classes, the relationship is not fully defined, and therefore the relationship is inconsistent. If a class has no relationships, the elements in that class cannot be related to any elements in the system. Because the existence of free-floating elements in a system defeats the purpose of complete system traceability, classes without relationships are defined to be inconsistent.

V. DISCUSSION

The Requirements Analyzer and Traceability Manager system provides extensive support for the system definition process. In particular, the system database provides complete documentation and traceability for engineering design. However, there are numerous extensions required to make this a complete system engineering tool. Because the tool has been designed in a modular format and complies with the object-oriented paradigm, additional capabilities can easily be added.

First, though this tool is tailored to requirements analysis, it does not include an automatic requirements parser. Though such parsers, used to automatically extract requirements from a document, are not perfect, they do allow the system engineer to simply guide the process instead of manually extracting all requirements. Therefore, this is an important an often requested capability for a requirements tool.

Second, there is no facility to define and explicitly support functional decomposition and analysis. In particular, there are no diagrams to help define the required flows. A minimum solution to this problem would be to add the standard Function Flow Block Diagram (FFBD) for functional decomposition and the N^2 diagram for data flow as put forth in the Systems Engineering Management Guide [10]. However, an integrated diagram which combines both data flow and functional flow should also be added in order to improve the system definition process.

Third, there are a number of other system engineering views that can play an integral role in the system engineering process. In particular, many engineers require the IDEF0 and Data Flow Diagrams (DFD) either for their own use or for purposes of reporting their results. Although the information on these diagrams is displayed in other views, the IDEF0 and DFD views should be implemented to support the engineers that use these views.

Fourth, a set of templates to generate standard reports and a custom report generator should be added. One of the major items on the engineer's desk top is the set of reports that document the system design. In particular, engineers need a number of standard reports such as the Data Item Dictionary, the DoD Standard 2167A, and Mil Standard 490 system and prime item level specification. Furthermore, custom reports are often required to document a specific piece of design information. Though the reports could be generated by developing a bridge to major publishing tools, an integrated report generator would be preferred so that it would not be necessary to freeze system development in order to produce the reports.

Finally, though provisions are made for statically checking the database for completeness, the ability to perform dynamic verification is totally absent. Any complete system engineering tool requires executable models so that the design can be both statically and dynamically verified. This capability, combined with the facility to define system flows, is the most important addition required for a complete system engineering tool.

Though all of the aforementioned capabilities are important for a complete system engineering tool, many are beyond the scope of the RATM system. In fact, it is not clear if it would be a good idea to combine all of these ideas into a single tool. Not all engineers will need all tools, and an engineering tool combining the complete set of capabilities would be too large, complex, and expensive. However, the ideal solution would be to develop all of these capabilities in a series of tools that could be integrated in order to provide each engineer with the tools that he or she needs.

VI. CONCLUSIONS

The importance of the system engineering process has grown as systems have become more complex and more systems require imbedded computer subsystems (primarily for the purpose of control). In order to design and develop effective and efficient systems within the constraints of today's problems, much more attention must paid to the system definition process. It is during system definition that most of the decisions determining the configuration of the system are made. Hence, good system definition is the cornerstone upon which solid and cost-effective systems are built.

The Requirements Analyzer and Traceability Management (RATM) system defined and developed during this project assists the engineer during system definition by capturing design information and maintaining complete design traceability. An object-oriented approach has been selected, because it is efficient, compact, and yields increased reliability. The tool is coded in Smalltalk, an excellent prototyping language which supports the object-oriented approach and is available on small computer platforms. Peter Chen's Entity-Relationship-Attribute database structure is implemented to fully describe the system being defined. However, database structure alone is meaningless. Therefore, a set of "user-friendly" interfaces consisting of a combination of text and graphics views that are optimized to fulfill the system engineer's needs are included. In particular, industry standards such as the hierarchy diagram and the text view as well as the system engineering editor are included. Furthermore, a requirement extractor is implemented in order to facilitate the specification of originating requirements. Because projects often have specific needs and there may be unanticipated needs for stating requirements, the basic database structure itself is extensible via a schema extender. Finally, the tool is expandable and extendable in order to serve as the basis of a tool to more completely satisfy the needs of system engineers. Though this tool has been implemented, the code is not released, because it is proprietary information. The completed tool is expected to be released as a commercial product to support the system definition process.

Though this tool is not yet a complete engineering tool, it does provide considerable assistance to the engineer during the definition process. By automating many of the routine tasks, this tool eliminates many of the errors which result from inconsistencies that naturally occur when trying to manually maintain multiple models of a single system. Furthermore, the tool's implementation is designed so that the tool will fit on "everyday" computers that engineers use (the IBM PC and Apple Macintosh). Therefore, the tool will be widely accessible. Though the tool developed will serve as the basis of a complete system definition tool, the Requirements Analyzer and Traceability Management tool provides much needed assistance to system engineers by supporting the system definition process, thereby filling the need for a "user-friendly" support tool that is accessible to most system engineers.

VII. SUMMARY

The Requirements Analyzer and Traceability Manager (RATM) designed, developed, and implemented is a prototype computer-aided system engineering tool to support the system definition process with an emphasis on requirements analysis and traceability management. RATM is implemented using an object-oriented approach and is prototyped in Smalltalk. From system model requirements, Peter Chen's Entity-Relationship-Attribute database structure representing elements (entities), pertinent information about the elements (attributes), and their interactions (relationships) is selected to maintain a complete, consistent system definition. The tool, designed for engineers, has "user-friendly" interfaces to allow efficient manipulation and representation of the data in the system definition. These interfaces include a system engineering editor (the primary interface for browsing and modifying the database), two primary engineering views (the text view and hierarchy view) to provide diagrams currently used by system engineers, a requirement extractor to aid the engineer in requirement specification, and a schema extender to allow the user to tailor the database structure to respond to specific project needs. RATM is an expandable, extendable prototype to permit future upgrades to support additional engineering views, standard reports required by engineers, and dynamic verification of models. At a time when system definition has become increasingly important in designing efficient, error-free, cost-effective systems, this computer-aided system engineering tool provides an accessible source of automated assistance to support the engineer.

VIII. LITERATURE CITED

[1] Benjamin S. Blanchard and Wolter J. Fabrycky, System Engineering and Analysis, 2nd ed., Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1990.

[2] Thomas E. Bell, David C. Bixler, and Margaret E. Dyer, "An Extendable Approach to Computer-Aided Software Requirements Engineering", IEEE Software, January 1977.

[3] L. Baker, Distributed Computing Design System: A Technical Overview, CDRL B003, TRW System Development Division, Huntsville, Alabama, July 1987.

[4] Smalltalk/V Mac: Tutorial and Programming Handbook, Digitalk, Inc., Los Angeles, 1989.

[5] Adele Goldberg, Smalltalk-80: The Interactive Programming Environment, Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

[6] Jim Diederich and Jack Milton. "Experimental Prototyping in Smalltalk", IEEE Software, May 1987.

[7] Jakob Nielson and John T. Richards, "The Experience of Learning and Using Smalltalk", IEEE Software, May 1989.

[8] Peter Chen, The Entity-Relationship Approach to Logical Data Base Design. Q.E.D Information Sciences, Inc., Wellesley, Massachusetts, 1977.

[9] Long, Dinsmore, Spadaro, Alford, et al., "The Engagement Logic and Control Methodology as Derived, Defined, and Applied at TRW", unpublished notes, 1968-1972.

[10] Systems Engineering Management Guide, Defense Systems Management College, Washington, D.C., 1989.

APPENDIX A: RATM FEATURE SUMMARY

The Requirements Analyzer and Traceability Manager (RATM) system consists of five integrated modules (element editor, text view, hierarchy diagram, requirement extractor, and schema extender) that have been built upon a common interface. This interface is based upon the standard Macintosh window environment and follows all Macintosh standards in order to minimize the learning curve for the interface.

Because the standard interface has been tailored to maximize the efficiency of the RATM environment and because many users may be unfamiliar with the Macintosh interface, the RATM environment will be discussed. Then, each of the five primary modules of the system will be discussed. All features, commands, and shortcuts for each module will be briefly outlined.

1. The RATM Environment

Though the RATM system consists of several different modules, the integrated tool shares a common environment. This environment is comprised of a standard, "user-friendly" Apple Macintosh interface. Because the interface is built around windows and menus and is mouse driven, very little time and effort is required for the engineer to learn to use the system effectively. Furthermore, any mistakes made by the user are met by clear, non-destructive error messages, so that the user understands the problem but is not harmed by the mistake. Thus, the RATM system is built on an extremely convenient, informative, safe user interface.

Because many users may not be familiar with the basic operations of windows and menus, the standard Macintosh interface of windows and menus will be discussed along with any modifications made for the RATM system. Then, because the modules of RATM share several menus, these common menus and their commands will be discussed. Finally, the user preferences window which allows the user to establish defaults for the entire system is discussed.

1.1 RATM Environment Basics

The basic RATM environment is built around the standard Macintosh environment interface of windows and menus. Though the windows and menus conform to all Macintosh standards, a brief description of both has been included for the benefit of those users who are unfamiliar with the Macintosh user interface. Furthermore, because the standard Macintosh cursor appearance has been enhanced to communicate information about the system status, a summary of the RATM cursor behavior has also been included.

1.1.1 Windows

In the RATM environment, windows are used to display system information in a variety of formats. Though each window may contain slight variations such as multiple panes versus a single pane, all windows are built upon the same basic model. Figure A.1 shows a standard RATM window and all of its components. Though many windows have all of the components shown in Figure A.1, several custom windows have been designed in such a way that some of the standard window components would be meaningless. For instance, the user preferences window has a fixed size, because the contents of the window are fixed (and not having to resize the window saves the user time). Therefore, the scroll bars and grow box have been removed from the user preferences window, because they would not add any functionality. The standard components and their functions conform to Macintosh standards.

- Close Box

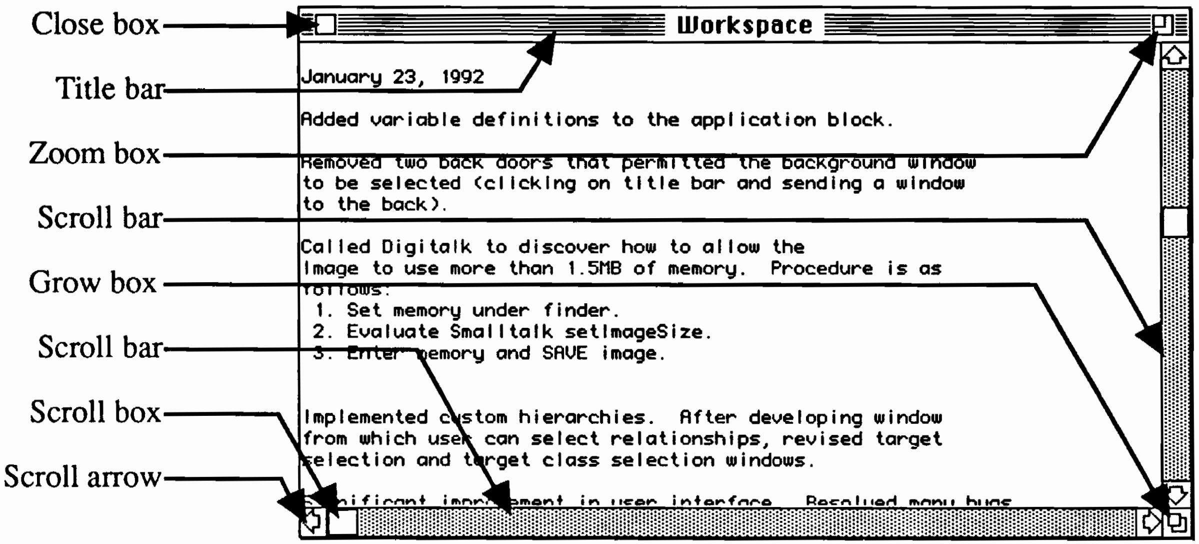
  When the user clicks in the close box of the active window, the window closes and disappears. If the window contains a text pane which has been modified but not saved, the user will be prompted whether or not to save the changes before the window is closed.

- Zoom Box

  When the user clicks in the zoom box, the active window expands to fill the entire screen. The user can return the window to its original size and position by clicking in the zoom box again.

- Scroll Box

  The user can scroll either horizontally or vertically by clicking on the appropriate scroll arrow. The position of the scroll box indicates approximate location in the window's contents (i.e., in Figure A.1, the text displayed in the window is approximately one-third of the way down in the window's contents). Many windows or panes that are scrollable do not have scroll bars (e.g., the relationship pane in the element editor). To scroll in these windows, the user should hold down the option key and



January 23, 1992

Added variable definitions to the application block.

Removed two back doors that permitted the background window
to be selected (clicking on title bar and sending a window
to the back).

Called Digitalk to discover how to allow the
Image to use more than 1.5MB of memory. Procedure is as
follows:
1. Set memory under finder.
2. Evaluate Smalltalk setImageSize.
3. Enter memory and SAVE image.

Implemented custom hierarchies. After developing window
from which user can select relationships, revised target
selection and target class selection windows.

Significant improvement in user interface. Resolved menu bugs

Figure A.1 Components of a Window

click and hold in the window until the scroll cursor appears.

- Grow Box

  By clicking and dragging in the grow box (holding down the mouse button), the user can resize the window. Once the window is the desired size, the user should release the mouse button. The user can return the window to its minimum size by pressing the option key and clicking in the grow box.

The only limiting factor on the number of windows that the user can have open at a time is the amount of memory in the user's computer. Because multiple windows can be open at the same time, certain conventions are required to indicate which window is currently active. Therefore, the active window is displayed in front of all other windows on the screen (i.e., it is on top of the stack). Furthermore, the title bar of the active window has parallel black stripes as shown in Figure A.1. All other windows have white title bars. To activate a window, the user should place the cursor inside the window and click the mouse button. The window will be brought to the top of the stack and become active.

The RATM environment contains two different window types, standard and modal. If a standard window is active, the user can switch between this window and any other window. These windows have title bars with parallel black stripes as shown in Figure A.1. However, some windows have solid black title bars. These windows, along with dialog boxes that prompt the user to enter specific information, are modal windows. Because of the circumstances under which a modal window or dialog is opened, the action prompted for by that window must be completed and the window closed before the user can activate another window. Therefore, while a modal window is active, clicking anywhere beyond the limits of the window results in the workstation bell being sounded to indicate that the current window must be closed before another window can be selected.

1.1.2 Menus

While the RATM system uses windows to display information, menus are used to allow the user to issue a command. Some menus contain choices that are always valid (e.g., the File Menu), and other menus are applicable to a single window only (e.g., the Editor Menu). The titles of all menus associated with the active window are displayed in the menu bar at the top of the screen as shown in Figure A.2. All menus are standard Macintosh pull-down menus and are normally dormant (unseen). As illustrated in Figure A.2, a menu becomes visible when its title is clicked on.

All menus in the RATM environment conform to the standard Macintosh conventions which are as follows:

- Show a menu's contents by clicking on its title in the menu bar.
- Select a command in a menu by clicking on the menu title and holding down the mouse button and moving the cursor to the desired command. When the desired command is highlighted, release the mouse button.
- Invalid menu selections are shown in a light gray as shown in Figure A.2. These are commands that currently cannot be carried out because certain conditions are not met (i.e., adding a target when no relationship to add a target to has been selected).
- Keyboard select a command by pressing the command key and the keyboard equivalent for the given command (if the command has a keyboard equivalent). In the case of the **Create Element** command shown in Figure A.2, the command can be selected by pressing the "E" key while holding down the command key (as notation, this is shown as Command + E).
- Close a menu by moving the mouse outside of the boundaries of the menu and releasing the mouse while no command is highlighted.

1.1.3 Cursor

RATM uses nine different cursor shapes to communicate the system status to the user. For example, the eye cursor indicates that the a file is being read from disk, and the pencil indicates that a file is being written to disk. The complete list of cursors displayed by the RATM system and the meanings of the cursors are shown in Figure A.3.

1.2 System Menus

The many integrated modules of the RATM software share several menus. These menus contain system commands that are either always valid (e.g., **Save Image** in the File Menu) or commands that apply to most windows (e.g., **Send to Back** in the Window Menu). The following menus are shared by most system modules: File, Edit, Window, and Utilities.

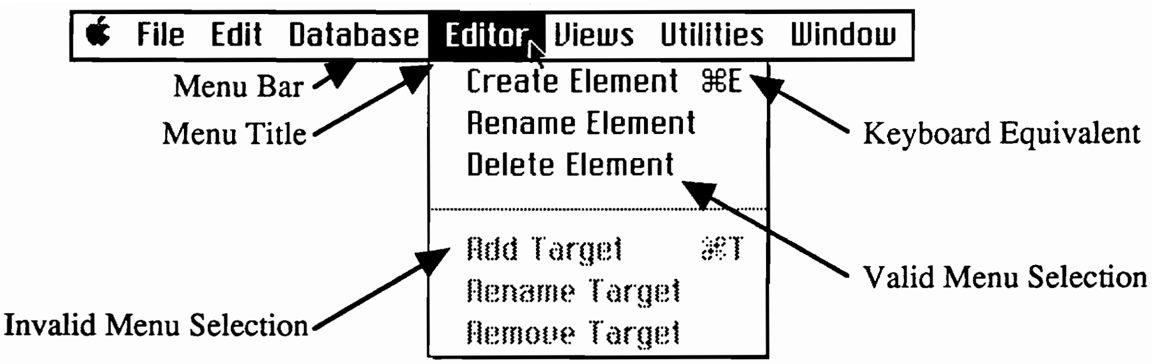


Figure A.2 Menu Bar and Editor Menu

Icon	Name	Description
┘	Corner	Displayed when framing rectangular areas
⌚	Execute	Displayed when execution is taking place
☜	Finger	Displayed when selecting an area on a diagram
✋	Hand	Displayed when dragging a diagram to change the display area
↖	Normal	Standard cursor
┌	Origin	Displayed when framing rectangular areas
◁	Read	Displayed when reading from a file
✥	Scroll	Displayed when scrolling a pane in one of the four directions
▤	Write	Displayed when writing to a file

Figure A.3 Cursor Summary

1.2.1 File Menu

The File Menu contains commands that allow the user to interact with the Macintosh operating system to manipulate files. This menu includes all standard file commands for Macintosh applications as well as special commands for the RATM application. The commands in the File Menu are: **New**, **Open**, **Save**, **Save as, Page Setup, Print, Save Image, Save Image as,** and **Quit**.

1.2.1.1 New (Command + N)

The command **New** allows the user to open a new, blank text file that can be saved to disk. The window (titled "Workspace") is intended as a scratchpad which can be saved and used with other applications.

1.2.1.2 Open (Command + O)

The command **Open** allows the user to open a text file. The user is prompted to select a file from the standard Macintosh dialog for opening a file. Once open, the file can be edited and copied to or from.

1.2.1.3 Save (Command + S)

The command **Save** performs one of two different functions depending upon the type of the active window. If the active window is a file editor (opened via the **Open** or **New** command), the **Save** command saves the contents of the active window to the disk. If the active window consists of multiple panes (e.g., the element editor, text views, requirement extractors), this command forces the active text pane to accept any modified text that has been entered into the pane.

1.2.1.4 Save as

The command **Save as** allows the user to save the contents of the currently active window to a file name specified by the user via a standard Macintosh file dialog.

1.2.1.5 Page Setup

The command **Page Setup** allows the user to set the various printer options via the standard Macintosh dialog box. These print options include paper size, enlargement or reduction, and page orientation (landscape or portrait).

1.2.1.6 Print (Command + P)

The command **Print** allows the user to print the contents of the active window to the printer, if the active window is a text editor (e.g., notebook, file editor). The user can specify the number of copies and print options via the standard Macintosh print dialog.

1.2.1.7 Save Image

The command **Save Image** allows the user to save the image file which contains both the tool and the data in a binary format under its current name.

1.2.1.8 Save Image as

The command **Save Image as** allows the user to save the image file under a new name specified by the user via a standard Macintosh file dialog.

1.2.1.9 Quit (Command + Q)

The command **Quit** allows the user to exit the tool. Before exiting the tool, the user is given a chance to save the image file or return to the tool without exiting.

1.2.2 Edit Menu

The Edit Menu contains the standard Macintosh commands used to manipulate text. Therefore, this menu is displayed for all windows in which text can be modified (i.e., all panes in the RATM system with the exception of the hierarchy diagram window). The commands in the Edit Menu are: **Undo**, **Cut**, **Copy, Paste, Clear, Select All, Find, Replace**, and **Search Again**.

1.2.2.1 Undo (Command + Z)

The command **Undo** allows the user to undo the last edit operation performed in the active window. Actions which can be undone include insertions, deletions, cuts, and pastes.

1.2.2.2 Cut (Command + X)

The command **Cut** deletes the selected text from the active text pane. The deleted text is placed on the clipboard. The text can then be pasted into a text pane via the **Paste** command.

1.2.2.3 Copy (Command + C)

The command **Copy** copies the selected text from the active text pane to the clipboard. The text can then be pasted into a text pane via the **Paste** command.

1.2.2.4 Paste (Command + V)

The command **Paste** inserts the text from the clipboard into the active text pane at the insertion point. If text is selected when this command is invoked, the selected text is replaced by the text from the clipboard.

1.2.2.5 Clear

The command **Clear** deletes the selected text in the active text pane. Unlike the **Cut** command, the text is not placed on the clipboard and is therefore lost.

1.2.2.6 Select All (Command + A)

The command **Select All** selects all text in the active text pane. The selected text can then be cut, copied, or deleted.

1.2.2.7 Find (Command + F)

The command **Find** allows the user to search the active text pane for a text string. The user is presented with a dialog which allows the user to specify the text string and the direction of the search (forward or backward).

1.2.2.8 Replace (Command + R)

The command **Replace** allows the user to replace all occurrences of a specified text string with a new text string. The user specifies the old string, the new string, and the conditions under which the old string should be replaced by the new string.

1.2.2.9 Search Again (Command + G)

The command **Search Again** allows the user to perform the last search operation in the direction previously specified in the active text pane.

1.2.3 Utilities Menu

The Utilities Menu contains commands that allow the user to access the many system utilities built into the RATM software. These commands include: **Open**

Requirement Extractor, Open Schema Extender, Open Notepad, and **User Preferences.**

1.2.3.1 Open Requirement Extractor

The command **Open Requirement Extractor** allows the user to open a requirement extraction window. The user is prompted to select the desired document from a standard Macintosh file dialog of all text files. A requirement extractor is then opened on the specified document. Because the requirement extractor is a tool to used to enter data into the system database, this command is not on the Utilities Menu when the database structure is being modified via the schema extender.

1.2.3.2 Open Schema Extender

The command **Open Schema Extender** allows the user to change to the schema extension facility to modify the current schema. In order to manipulate the schema, the database must be empty. Therefore, if there are any elements in the database, the user is warned that opening the extender will erase all data. If the user wishes to proceed, all windows are closed, and the schema extender is opened. Obviously, this command is not on the Utilities Menu when the schema extender is open.

1.2.3.3 Open Notepad

The command **Open Notepad** allows the user to open the system notepad to use as a scratch pad for temporary notes. If the notepad is already open, its window is brought to the front of the display instead of another window being opened.

1.2.3.4 User Preferences

The command **User Preferences** opens a modal user preferences window to allow the user to set the basic options for the system database and views.

1.2.4 Window Menu

The Window Menu contains commands that allow the user to change the active window's size, fonts, and location. These commands include: **Send to Back, Collapse, Zoom, Change Text Font, Change List Font., Redraw Window**, and **Stack Windows.**

1.2.4.1 Send to Back

If more than one window is currently open, the command **Send to Back** allows the user to send the active window to the back of the stack of RATM windows.

1.2.4.2 Collapse

The command **Collapse** allows the user to show only the title bar of the active window (thereby reducing the amount of screen clutter without closing a window). The title bar can then be moved and positioned as desired. To reopen the window to its previous size, the user can click in the zoom box. Option-clicking in the zoom box is a shortcut for the collapse method.

1.2.4.3 Zoom

The command **Zoom** allows the user to toggle size of the active window between its present size and the size of the entire screen. This can also be accomplished by clicking in the zoom box of the window.

1.2.4.4 Change Text Font

The command **Change Text Font** allows the user to set the font and font size for text panes in the active window and all future windows through a dialog box with two pop-up menus. Pre-existing windows that are not active are not affected.

1.2.4.5 Change List Font

The command **Change List Font** allows the user to set the font and font size for list panes in the active window and all future windows through a dialog box with two pop-up menus. Pre-existing windows that are not active are not affected.

1.2.4.6 Redraw Screen

The command **Redraw Screen** causes the system to redraw all open windows. A shortcut for this command is pressing Command + \

1.2.4.7 Stack Windows

The command **Stack Windows** results in the system rearranging all open windows into a neat, staggered stack for easy access to all windows.

1.3 User Preferences

The user preferences window is a modal interface that allows the user to change the basic settings for the entire system. The window consists of multiple fields and buttons as shown in Figure A.4. Those system parameters which can be manipulated through the user preferences window can be broken into two basic categories: general options and diagram display options.

1.3.1 General Options

The general options set through the user preferences window are the system author shown at the top of the window and the parameters controlled by the buttons at the bottom of the window. The author for all future elements is show in the author field. In the case illustrated by Figure A.4, any future element created will have the value "System User" as its author. Changing the author field does not affect elements that have already been created.

Though the author parameter can assume any value, most system parameters can assume only one of two values. Therefore, these parameters are set by toggling one of two buttons on. The buttons are as follows:

- Auto Accept

  If the auto accept feature is turned on, text entered in text panes on the element editor and text views will be saved automatically. If the feature is off, changes made to those panes must be saved via the **Save** command.

- Sort Preference

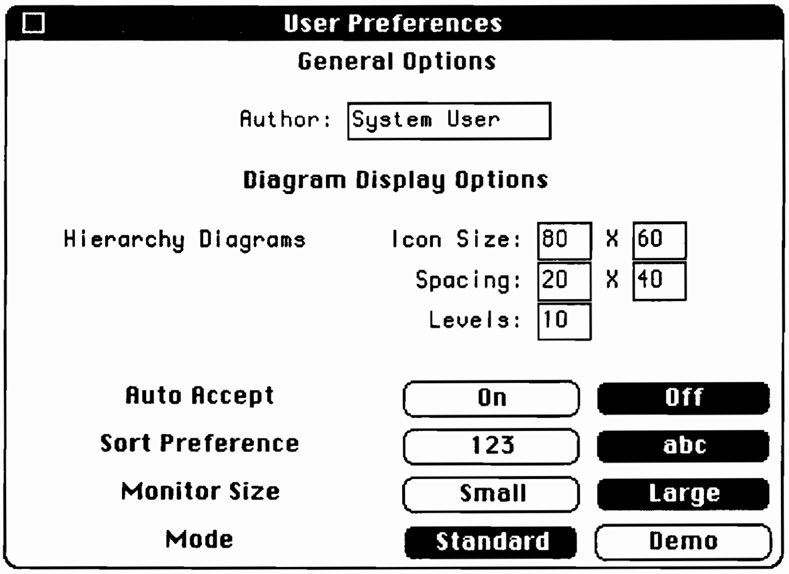
  The sort preference parameter determines the default method to be used to sort all element lists. If the button labeled "123" is pressed, element lists will be sorted by the element number. If the button labeled "abc" is pressed, element lists will be sorted alphabetically.

- Monitor Size

  The monitor size parameter helps determine the default size and placement of many windows (e.g., Add Target Selection window). If a standard twelve inch monitor is being used, the parameter should be set to small. However, if a nineteen inch monitor is being used, the parameter should be set to large to obtain the best window placement.

- Mode

  The mode parameter allows the user to switch between standard and demo



User Preferences

General Options

Author: System User

Diagram Display Options

Hierarchy Diagrams

Icon Size: 80 X 60

Spacing: 20 X 40

Levels: 10

Auto Accept: On [✓ Off]

Sort Preference: 123 [✓ abc]

Monitor Size: Small [✓ Large]

Mode: [✓ Standard] Demo

Figure A.4 User Preferences Window

modes. The only time that this button should be changed is if changes must be made to the basic system code. To access and update the system code, the user must set the mode to demo and close the user preferences window. The user is then prompted to enter the system password. If the password is correct, the tool is closed, and changes may be made to the system code. Otherwise, the tool will remain in standard mode.

1.3.2 Diagram Display Options

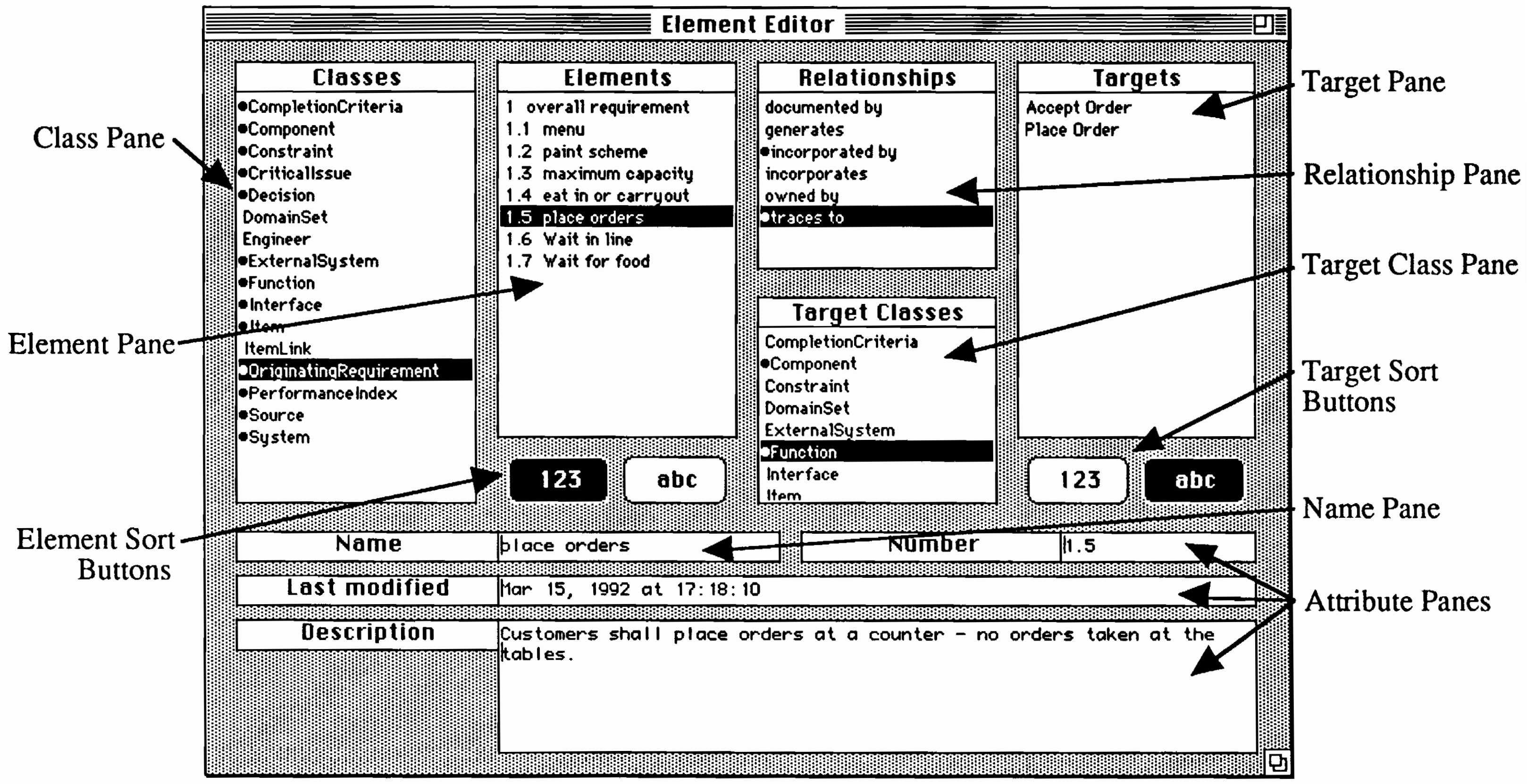
The diagram display options set through the user preferences window affect any diagrams opened after the parameters are changed and the window is closed. Any diagrams already open will not be affected by changes to the diagram display options. To change the options for a diagram currently open, the user must use the **Display Options** command for the given diagram.

The parameters that can be changed through the user preferences window include icon size, icon spacing, and the initial number of levels for each diagram. The user can specify the height and width of the icon and the vertical and horizontal spacing between the icons (all in pixels) by simply entering a number in the appropriate field and pressing return. The initial number of levels to be displayed on the diagram can also be specified so that the user can produce a diagram containing as little or as much detail as the user desires.

2. Element Editor

The element editor, the primary interface for browsing and updating the system database, consists of nine different panes divided into two basic groups as shown in Figure A.5. The five upper panes are list panes showing elements, classes, and relationships from which the user can select simply by clicking on the name in the list that is desired. The lower four panes are labeled text panes displaying the name and three attributes of the selected element.

In the first list pane, the classes (e.g., Component, Function) are displayed. When the user selects the desired class, the elements within that class are shown in the elements pane. In Figure A.5, the class OriginatingRequirement has been selected resulting in the requirements in the database being displayed in the element pane. The relationship pane lists all possible relationships for the selected class. When the user selects a relationship,



Name: place orders

Number: 1.5

Last modified: Mar 15, 1992 at 17:18:10

Description: Customers shall place orders at a counter - no orders taken at the tables.

Figure A.5 Element Editor

all possible target classes are listed in the target classes pane. In Figure A.5, the relationship "traces to" is selected, and the possible target classes (e.g., CompletionCriteria, Component, and Constraint) are displayed. Finally, when the user selects a target class, the targets of that relationship in the selected target class are displayed. In the case shown in Figure A.5, the relationship has two targets in class Function , "Accept Order" and "Place Order".

In the class pane shown in Figure A.5, the class OriginatingRequirement is preceded by a black circle (•). This black circle indicates that the database includes elements of type OriginatingRequirement. If a relationship is preceded by a black circle, the selected element is linked to at least one target by the relationship. Furthermore, if a target class is preceded by a dot, there is at least one target in the target class.

The selection of a specific element in the element pane causes the attributes corresponding to that element to be displayed in the lower text panes. The first pane displays the name of the selected element. The remaining three panes display attributes of the selected element. The attributes displayed in Figure A.5 are "Number", "Last modified", and "Description". To change the attribute displayed in a specific pane, the user can click on the label for the pane. A menu containing the five standard attributes for every element ("Abbreviation", "Author", "Created on", "Last modified", and "Number") will pop-up, and the user can select the desired attribute. Only standard attributes can be displayed and edited on the element editor.

Not only do the lower four pane display the element name and attributes, but they are also fully operational text-editing windows. Text can be added, deleted, cut, pasted, and copied through a standard Apple Macintosh interface. However, the author, created on, and last modified attributes are non-editable attributes. This means that the user cannot explicitly enter values for these values. The value of the "Author" attribute is read from the user preferences at the time that the element is created. When the element is created, the "Created on" attribute is also specified. Neither of these two attributes can be changed. On the other hand, the "Last modified" attribute is automatically updated whenever a change is made to the element definition (either an attribute value is changed or a target is added or removed).

Though the attributes of the selected element can be modified through the text panes on the element editor, the only method to manipulate elements and relationships is though the menus. The Database Menu and Editor Menu allow the user to manipulate the database. The Views Menu allows the user to open alternate views on the selected element.

2.1 Database Menu

The Database Menu contains all commands that deal with the entire database. These commands include: **Import Database, Export Database, Erase Database, Print Database Report**, and **Run Consistency Check.**

2.1.1 Import Database

The command **Import Database** allows the user to load a database file from disk into memory. The user is presented with a standard Macintosh dialog to select a text file to open. This command deletes any elements currently in the system. Therefore, the user is warned of this consequence before continuing.

2.1.2 Export Database

The command **Export Database** allows the user to write all element definitions in the database to a text file on disk. By exporting the database, the user can create a relatively small, stable backup to the database stored in the image file. Because the export includes only the element definitions, it requires only a fraction of the disk space of an image file which includes both the data and system code. Furthermore, since the export is a text file instead of a binary file, it is much less likely to become corrupted (since displacing a single bit in a text file will result only in a single character being changed whereas displacing a single bit in a binary file can corrupt all data in the file).

2.1.3 Erase Database

The command **Erase Database** causes the image to erase all elements in the database. Because of the potential loss of data, the user is asked to confirm the choice to erase all entities before any action is taken.

2.1.4 Print Database Report

The command **Print Database Report** allows the user to create a structured text report file on disk containing the definitions of all attributes and relationships for the elements in the database. The report can be opened via the **Open** command in the file menu and printed in order to obtain a hard copy. Furthermore, because the report is stored in a formatted text file, it can be read (and modified) by any word processor in order to be included in other reports.

2.1.5 Run Consistency Check

The command **Run Consistency Check** has not yet been implemented. When implemented, this command will allow the user to perform a predefined set of checks to determine if the elements currently in the database are incomplete.

2.2 Editor Menu

The Editor Menu contains the commands that allow the user to update the system database. These commands include: **Create Element, Rename Element, Delete Element, Add Target, Rename Target,** and **Remove Target**.

2.2.1 Create Element (Command + E)

The command **Create Element** allows the user to define a new entity in the selected class. The user is prompted to enter a name for the new element, and if the name does not conflict with the name of an element currently in the selected class, the new element is created with nil values for all attributes and relationships.

2.2.2 Rename Element

The command **Rename Element** allows the user to change the name of the selected element. The user is prompted to enter the new name for the element. If the name entered is not the same as a name of an element in the selected class, all instances of the element in the database are renamed. An element can also be renamed by entering a new name in the name field on the element editor.

2.2.3 Delete Element

The command **Delete Element** allows the user to remove all instances of the selected element from the database. Because all instances of the element will be removed, the user must confirm this choice before the element is deleted.

2.2.4 Add Target (Command + T)

The command **Add Target** allows the user to add multiple targets in the selected target class to the selected relationship and automatically creates the complementary relationships linking the targets and the selected element. A modal selection window consisting of a list of all elements in the target class and a set of five buttons (Add, Create, Remove, Done, and Cancel) as shown in Figure A.6 is opened.

When the selection window opens, all current targets of the selected relationship are preceded by a white circle (°). As elements are added to the target class list, their names are preceded by a black circle (•). An element can be added to the target list either by selecting the desired target and clicking on the Add button or by double-clicking on the desired element. If the user wishes to add an element that does not yet exist, the user can create a new element by clicking the Create button. If an element is inadvertently added to the target list, it can be removed by selecting the target and clicking Remove. However, pre-existing targets cannot be removed in this manner. They must be removed via the command **Remove Target.**

Once all desired element have been added to the target list, they can be added to the relationship by clicking the Done button. Note that no changes are made to the relationship until the Done button is pressed. Therefore, if the user wishes to close the window without changing the targets of the selected relationship, the user should click on the Cancel button.

2.2.5 Rename Target

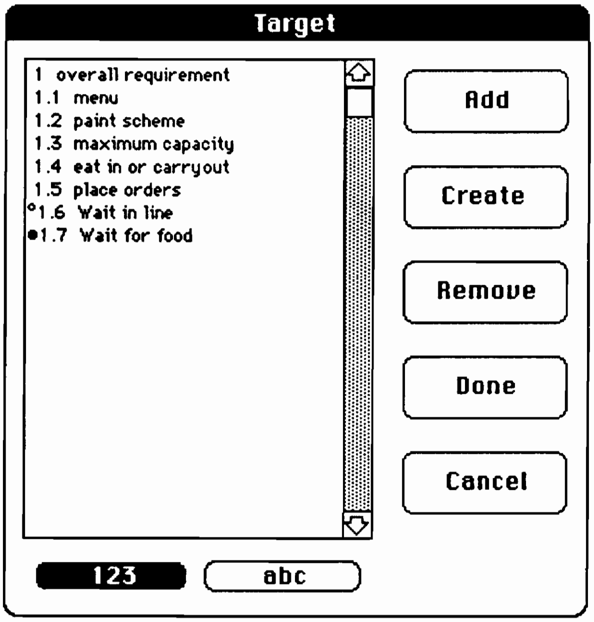
The command **Rename Target** allows the user to change the name of the selected target. This command is identical to the command **Rename Element** with the exception that the name of the selected target is changed instead of that of the selected element.

2.2.6 Remove Target

The command **Remove Target** allows the user remove a target from the selected relationship. The selected target is removed from the relationship, and the complementary relationship linking the selected element and the selected target class is removed.

2.3 Views Menu

The Views Menu and its sub-menu contain commands that allow the user to open one of the predefined views (either graphic or text) of the selected element. These commands include: **Text View**, **Hierarchy**, **Function**, **Physical**, **Traceability**, and **Custom**.



1 overall requirement
1.1 menu
1.2 paint scheme
1.3 maximum capacity
1.4 eat in or carryout
1.5 place orders
°1.6 Wait in line
●1.7 Wait for food

Figure A.6 Add Target Selection Window

This menu is also available from a text view. However, when the active window is a text view instead of the element editor, the commands open the specified view of the selected target instead of the selected element (which does not exist for a text view).

2.3.1 Text View

The command **Text View** opens a text view on the selected element that displays all attributes and relationships of the element. If a text view is already open for the selected element, that window is brought to the front of the screen instead of a new window being opened in order to avoid cluttering the screen unnecessarily.

2.3.2 Hierarchy

The menu selection **Hierarchy** opens a sub-menu of hierarchy views that may be opened on the selected element. Whichever hierarchy type is selected, a chart is opened with the selected element and all of its descendants as defined by the relationships defined by the hierarchy type. However, if a hierarchy diagram of the specified type is already open for the selected element, the window containing the chart is brought to the front of the screen instead of a new window being opened.

2.3.2.1 Function

The command **Function** opens a function hierarchy on the selected element which must be in class Function for the menu selection to be valid. The relationship used to define this hierarchy is "decomposed by". Because the function hierarchy is built on a single relationship, the links between elements on the hierarchy diagram are not labeled with the relationship name. Furthermore, because the relationship "decomposed by" only has one target (class Function), the default setting is to not display the element classes on the diagram.

2.3.2.2 Physical

The command **Physical** opens a physical hierarchy on the selected element which must be in class Component, ExternalSystem, or System for the menu selection to be valid. The relationship used to define this hierarchy is "built from". Because the physical hierarchy is built on a single relationship, the links between elements on the hierarchy diagram are not labeled with the relationship name. Furthermore, because the relationship "built from" only has one target (class Component), the default setting is to not display the element classes on the diagram.

2.3.2.3 Traceability

The command **Traceability** opens a traceability hierarchy on the selected element which must be in class Component, CriticalIssue, Decision, ExternalSystem, Function, OriginatingRequirement, Source, or System for the menu selection to be valid. The relationships used to define this hierarchy are "allocated to", "built from", "decomposed by", "documents", "generates", "incorporates", and "traces to". Because the traceability hierarchy is built on multiple relationships, the links between elements on the hierarchy diagram are labeled with the relationship name. Furthermore, because elements of different classes may be displayed, the default setting is to display the element classes on the diagram.

2.3.2.4 Custom

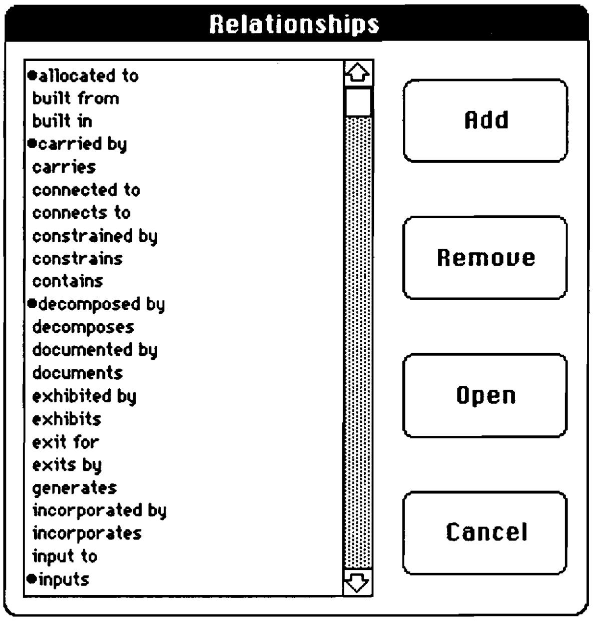
The command **Custom** opens a user-defined hierarchy on the selected element. The user is asked to select the relationships to define the hierarchy from a modal selection list of all relationships in the schema as shown in Figure A.7. Because the custom hierarchy may be built on multiple relationships, the links between elements on the hierarchy diagram are labeled with the relationship name. Furthermore, because elements of different classes may be displayed, the default setting is to display the element classes on the diagram.

As relationships are added to the selection list, their names are preceded by a black circle (•). A relationship can be added to the target list either by selecting the desired relationship and clicking on the Add button or by double-clicking on the desired relationship. If a relationship is inadvertently added to the selection list, it can be removed by selecting the relationship and clicking the Remove button.

Once all desired relationships have been added, the custom hierarchy can be opened by clicking the Create button. If the user wishes to exit without opening a chart, the Cancel button should be pressed.

3. Text View

All of the information on a particular element in the system database is displayed



●allocated to
built from
built in
●carried by
carries
connected to
connects to
constrained by
constrains
contains
●decomposed by
decomposes
documented by
documents
exhibited by
exhibits
exit for
exits by
generates
incorporated by
incorporates
input to
●inputs

Figure A.7 Relationship Selection Window

in the text view, a structured format illustrated in Figure A.8. The attributes are displayed in the upper portion of the window in labeled text panes. The relationships, target classes, and targets that complete the element definition are displayed in the lower portion of the window as shown in Figure A.8.

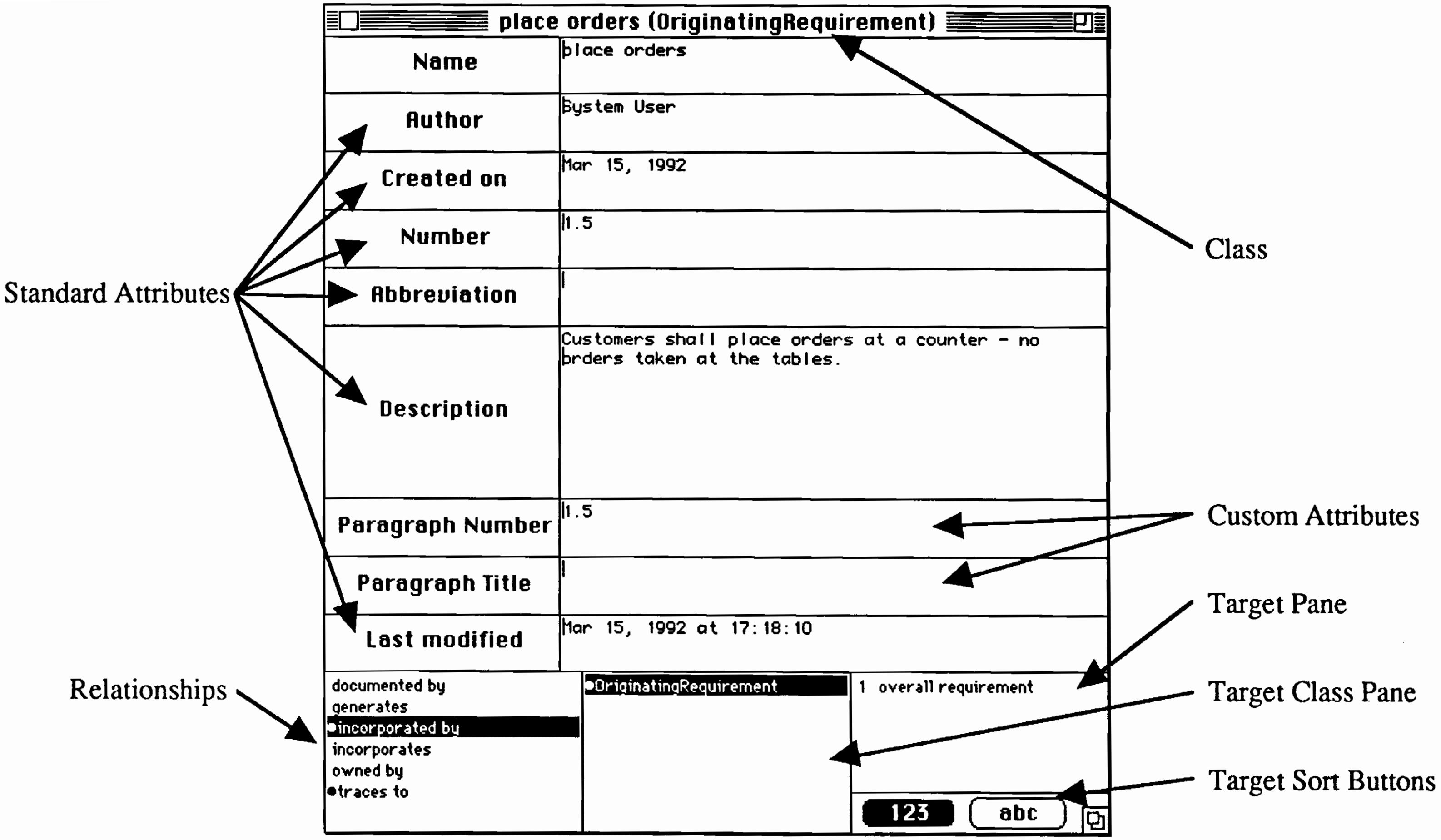
The first labeled field on the text view is the element name. By editing the contents of this pane, the user can rename the element. The remaining labeled text panes (up to fourteen panes) display the attributes for the element. As shown in Figure A.8, the attribute panes are displayed in the following order:

- Author (non-editable)
- Created on (non-editable)
- Number
- Abbreviation
- Custom attributes of the element in alphabetical order (up to 9)
- Last modified (non-editable)

The first four and the last attribute are all standard attributes for all elements in the system. All other element attributes are classified as custom attributes and are defined either exclusively for specific classes or are added to all classes by the system. Because of the nature of the custom attributes, they are not accessible from the element editor. Therefore, the only manner these attributes can be specified is through the text view. The sole exception to this is the case of OriginatingRequirements whose paragraph name and number can be specified when the element is created through the requirement extractor.

Three attributes ("Author", "Created on", and "Last modified") are non-editable attributes. Both the "Author" and "Created on" attributes are automatically determined at the time the element is created. In order to maintain complete design documentation this information must be maintained. The "Last modified" attribute is automatically set whenever a relationship or attribute is changed and assists in tracing modifications to the database.

Though the attribute panes in the text view are more extensive than those in the element editor, the list panes at the bottom of the text view are identical to those of the element editor. As in the case of the element editor, the user can select a relationship by clicking on the relationship name in the list pane. After a relationship is selected, the target classes for the selected relationship are shown in the target class pane. If there is only one target class for the selected relationship, it is automatically selected. Once a



place orders (OriginatingRequirement)

Name	place orders
Author	System User
Created on	Mar 15, 1992
Number	1.5
Abbreviation	
Description	Customers shall place orders at a counter - no orders taken at the tables.
Paragraph Number	1.5
Paragraph Title	
Last modified	Mar 15, 1992 at 17:18:10

documented by generates ●incorporated by incorporates owned by ●traces to	●OriginatingRequirement	1 overall requirement

Figure A.8 Sample Text View

target class is selected, all targets in that class are displayed in the target pane. The user can choose to sort the targets by name or element number by pressing the appropriate sort button as shown in Figure A.8.

Commands for the text view are contained in two menus. The Text View Menu contains commands to manipulate the relationships for the element. The Views Menu (covered in the discussion of the element editor) allows the user to open views on the selected target.

3.1 Text View Menu

The Text View Menu contains commands that allow the user to modify the ERA database definition of the element on which the text view has been opened as well as a command to output the text view to the printer. The set of commands for the Text View Menu is: **Add Target, Rename Target**, **Remove Target**, and **Hard Copy**.

3.1.1 Add Target (Command + T)

The command **Add Target** allows the user to add targets to the selected relationship. This command is identical the **Add Target** command in the element editor (discussed on page 54).

3.1.2 Rename Target

The command **Rename Target** allows the user to change the name of the selected target. This command is identical the **Rename Target** command in the element editor (discussed on page 56).

3.1.3 Remove Target

The command **Remove Target** allows the user to remove the selected target from the selected relationship. This command is identical the **Remove Target** command in the element editor (discussed on page 56).

3.1.4 Hard Copy

The command **Hard Copy** allows the user to output the current text view to the printer. A bitmapped version of the text view (identical to that of the text view as displayed on the screen) is sent to the printer, and a hard copy of the view is produced.

4. Hierarchy Diagram

The hierarchy diagram is a chart used to graphically display (but not modify) which elements are related to each other and what relationships constitute these links. Each diagram is defined by two primary parameters: a top-level element and a set of relationships. In computing the diagram, RATM starts with the top-level element and checks to see if any of the specified relationships have targets. If so, these targets are children of the top-level element and are displayed on the second level of the diagram. The children of the elements on the second level are then determined. This process continues until the elements on the lowest level of the diagram have no targets in the specified set of relationships. In Figure A.9, the top level element is "overall requirement". The targets of "overall requirement" are "eat in or carry out" and "place orders" (both by the relationship "incorporates"). In turn, "eat in or carry out" has four targets, and "place orders" has two targets. Though "Service Staff" is not a target of "overall requirement", it is said to be a descendent of "overall requirement" just as a grandchild is said to be a descendent of a grandparent.

The hierarchy diagram itself consists of two elements: icons and arcs. Each element on the chart is represented by an icon (also called a node). The icon contains both the name, number and class of the element (though the user can control whether or not the class is displayed). Furthermore, if the icon appears in multiple places on the diagram, a small black square is drawn in the upper-right hand corner of the icon as shown in Figure A.9. The arcs represents relationships between objects and targets. If the set of relationships for a hierarchy contains more than one relationship, each arc on the diagram is labeled to indicate which relationship links the two elements as shown in Figure A.9. However, when only a single relationship is used to define a diagram (as in the case of a Function Hierarchy), relationship labels are not displayed in order to keep the clutter on the diagram to a minimum.

In order to manipulate the appearance of the hierarchy diagram, the user must be able to select a given icon. This can be accomplished simply by clicking within the bounds of the desired icon. Furthermore, if the user wishes to select multiple icons at the same time, the user can perform the standard Macintosh shift clicks by holding down the shift key while clicking on the desired icons. The finger cursor can also be used to select multiple icons by positioning the cursor so that it is not on any icon and then holding down the mouse button and dragging the mouse until the desired icons are within the selection box.

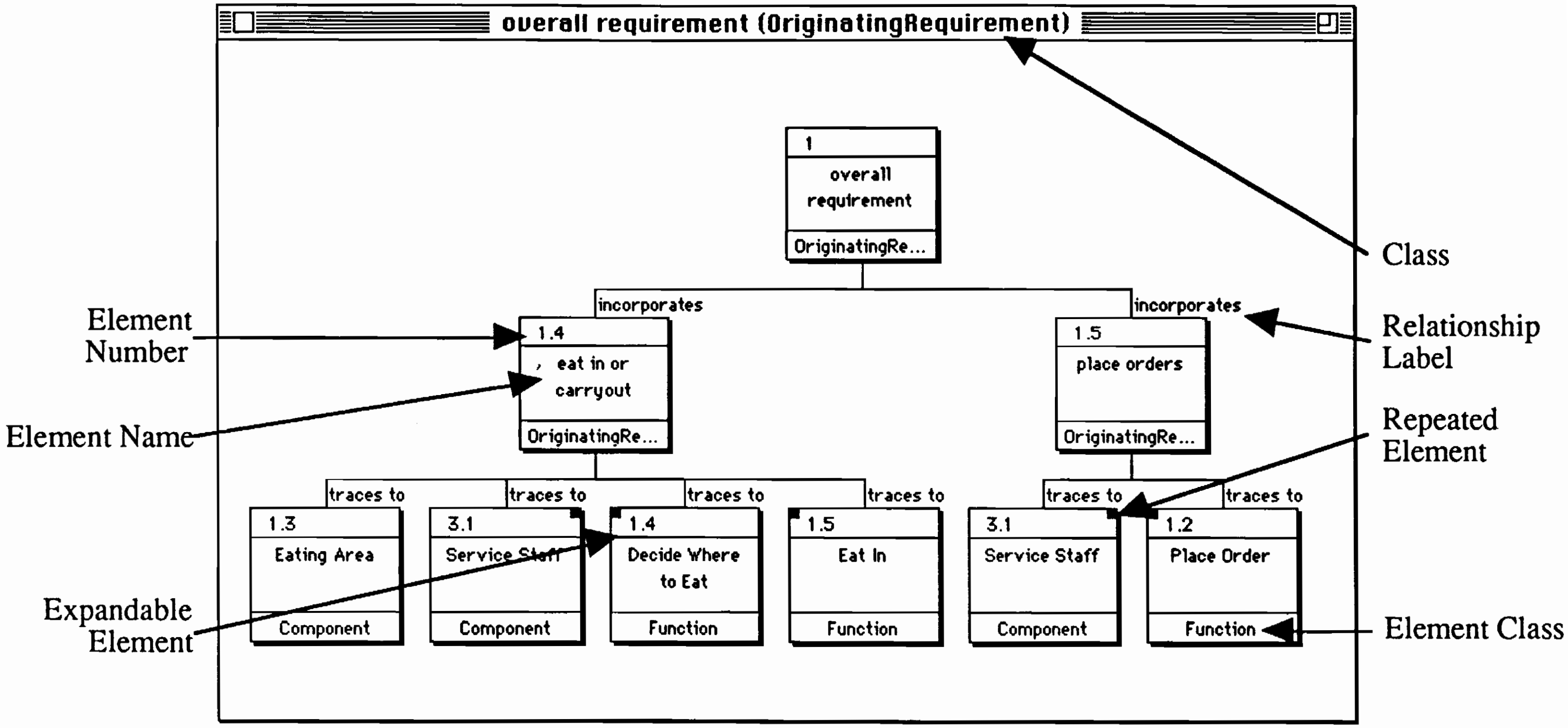


Figure A.9 Sample Traceability Hierarchy

The user can control the number of levels in the hierarchy diagram to reveal as much or as little detail as is required. Therefore, an icon can be collapsed to hide its descendants in which case the icons for the descendants are removed from the graph. An icon which has hidden descendants can be expanded to display its children. If the element has children which are not currently being displayed, a black rectangle is placed in the upper-left hand corner of the icon as shown in Figure A.9 to indicate that the icon is expandable.

In generating the hierarchy chart, several conventions are followed to make the diagrams as readable as possible. First, all children of an icon are sorted so that they appear in a standard order. The sort criteria (in the order of importance) are as follows:

- alphabetical by relationship
- alphabetical by class
- numerical by element number (those elements with numbers come before those elements without numbers)
- alphabetical by element name.

Second, in order to avoid recursive diagrams and diagrams that waste space by showing the same information more than once, the children of an element are shown only the first time the element is encountered. For instance, if in the process of searching for all descendants of the top level element the program encounters the element "Example" in class Function for the second time, the program will not check to see if "Example" has any children. Therefore, the children will only be displayed once. Because the hierarchy diagrams are computed from left to right, the leftmost occurrence of an icon will always show any children, and all occurrences of the icon will be marked to indicate that the icon appears elsewhere on the diagram as illustrated with the element "Service Staff" in Figure A.9.

Because the hierarchy diagrams only allow the user to display information in the system database instead of also modifying the database, only a single menu – the Hierarchy Menu – is needed to contain the necessary commands.

4.1 Hierarchy Menu

The Hierarchy Menu contains commands that allow the user to manipulate the appearance and content of the hierarchy view without modifying the database as well as a command that allows the user to generate a printed copy of the diagram. The commands on the Hierarchy Menu include: **Display Options, Expand, Collapse, Refresh Diagram, Recompute Diagram**, and **Hard Copy**.

4.1.1 Display Options

The command **Display Options** allows the user to change the diagram preferences for the current diagram only. This command opens a modal window consisting of multiple fields and buttons as shown in Figure A.10.

The user can specify the height and width of the icon and the vertical and horizontal spacing between the icons (all in pixels) by simply entering a number in the appropriate field and pressing return. The initial number of levels to be displayed on the diagram can also be specified so that the user can produce a diagram containing as little or as much detail as the user desires. Through the use of the buttons, the user can toggle whether or not the diagram should be framed with a border and if the element class should be displayed on each icon. As shown in Figure A.10, the user can also specify whether the hierarchy diagram should be centered horizontally or vertically when output to a printer. When the diagram preferences window is closed, the diagram is recomputed in accordance with the new preferences.

4.1.2 Expand (Command + E)

The command **Expand** allows the user to expand (show the descendants of) the selected elements. The user is prompted for the maximum number of levels to by which to expand the elements. After performing the expansions, the diagram is redisplayed. Double-clicking on an expandable element is a shortcut for expanding the element by a single level.

4.1.3 Collapse (Command + C)

The command **Collapse** allows the user to collapse (hide the descendants of) the selected elements. The elements are collapsed, and the diagram is redisplayed. Double-clicking on a collapsible element is a shortcut for this command.

4.1.4 Refresh Diagram

The command **Refresh Diagram** allows the user to redraw the window without recomputing the diagram. By refreshing the diagram, the user can clean up any garbage that is cluttering the diagram.

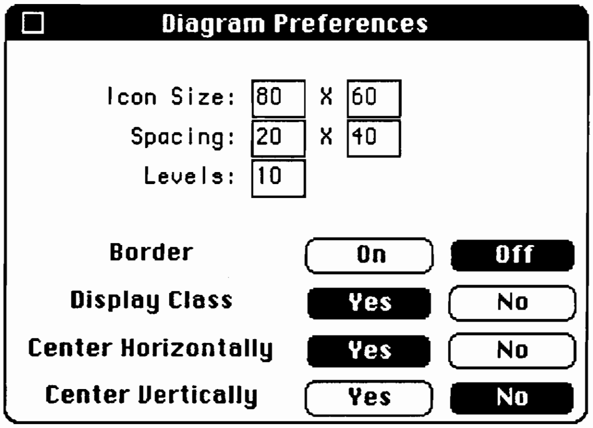


Diagram Preferences

Icon Size: 80 X 60

Spacing: 20 X 40

Levels: 10

Border On [✓ Off]

Display Class [✓ Yes] No

Center Horizontally [✓ Yes] No

Center Vertically Yes [✓ No]

Figure A.10 Diagram Preferences Window

4.1.5 Recompute Diagram

The command **Recompute Diagram** allows the user to recompute the diagram and redraw the display. By recomputing the diagram, the user can update the diagram to reflect any changes made to the database since the diagram was opened (such as an element being renumbered or renamed).

4.1.6 Hard Copy

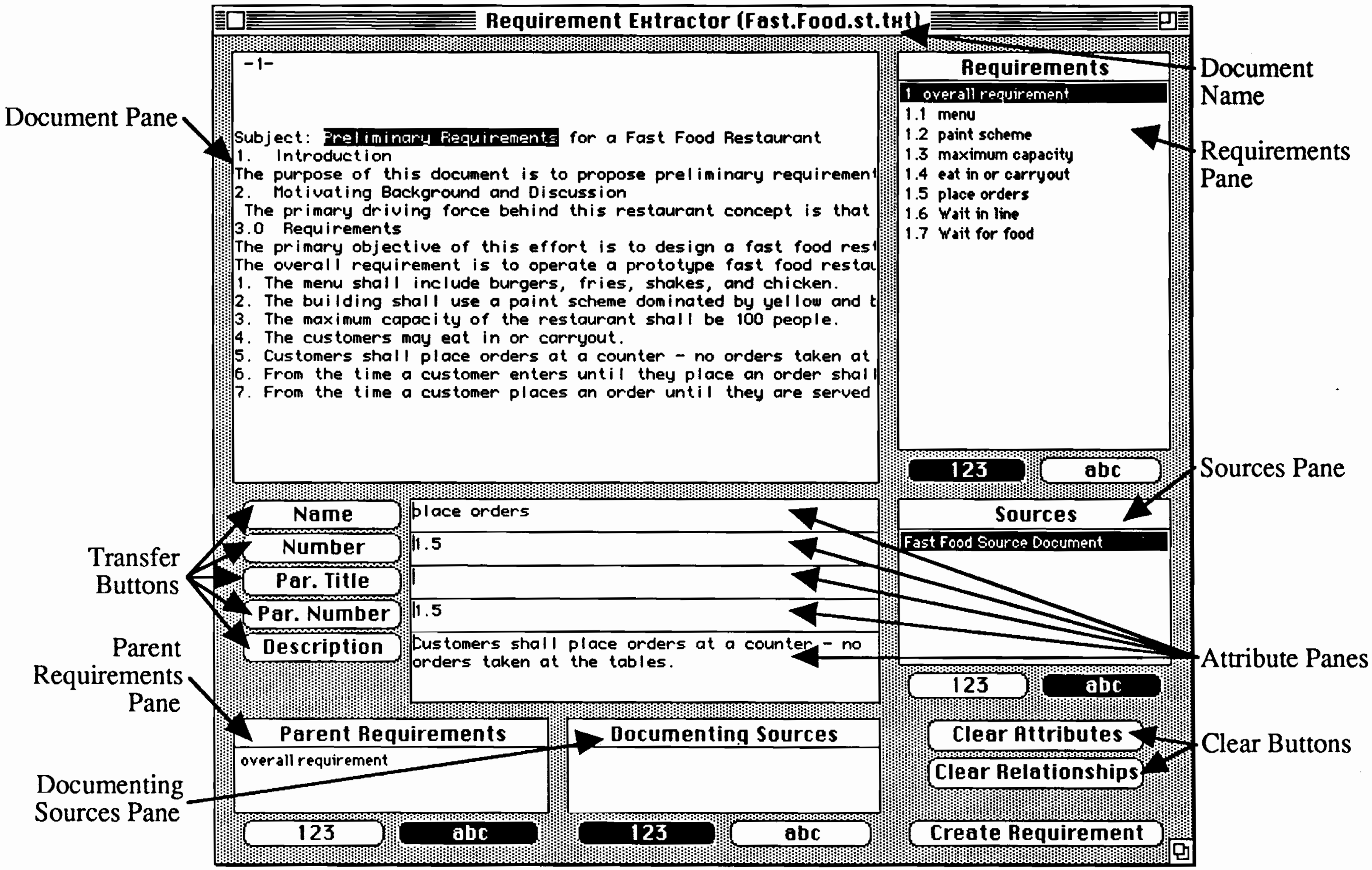
The command **Hard Copy** allows the user to produce a printed version of the diagram currently displayed on the screen. If the diagram is too large for a single sheet of paper, the diagram will be broken up into blocks printed on single sheets of paper which can then be placed together to show the complete diagram. The diagram may be centered horizontally or vertically and printed with or without a border depending upon the current settings for the diagram display options.

5. Requirement Extractor

Unlike the element editor, text view, and hierarchy diagram which can be used to browse the contents of the system database, the requirement extractor can only be used to add information to the system database. If the user has a document (in electronic format) containing the originating requirements for the system under consideration, the requirement extractor can be used to accelerate the process of entering the requirements into the system.

The requirement extractor window consists of ten panes and numerous buttons as shown in Figure A.11. First, the text pane in the upper-left hand portion of the window is the document pane. The five text panes directly below the document pane are the attribute panes. Below the attribute panes are the two list panes that display the targets for the relationships that can be specified in this window. Along the right side of the window are two list panes which display all elements in the OriginatingRequirement and Source classes.

When the user opens the requirement extractor on a specified file, the contents of the file are displayed in the document pane. Because this is a text pane, the user can highlight text or perform any other standard editing function. Most often, the user will simply want to extract segments of text from the document and enter it into the database. Therefore, transfer buttons have been implemented to facilitate this process. To move



Requirement Extractor (Fast.Food.st.txt)

```
-1-

Subject: Preliminary Requirements for a Fast Food Restaurant
1.   Introduction
The purpose of this document is to propose preliminary requiremen
2.   Motivating Background and Discussion
 The primary driving force behind this restaurant concept is that
3.0  Requirements
The primary objective of this effort is to design a fast food res
The overall requirement is to operate a prototype fast food restau
1. The menu shall include burgers, fries, shakes, and chicken.
2. The building shall use a paint scheme dominated by yellow and b
3. The maximum capacity of the restaurant shall be 100 people.
4. The customers may eat in or carryout.
5. Customers shall place orders at a counter - no orders taken at
6. From the time a customer enters until they place an order shal
7. From the time a customer places an order until they are served
```

Requirements

1 overall requirement
1.1 menu
1.2 paint scheme
1.3 maximum capacity
1.4 eat in or carryout
1.5 place orders
1.6 Wait in line
1.7 Wait for food

Name	place orders
Number	1.5
Par. Title	
Par. Number	1.5
Description	Customers shall place orders at a counter - no orders taken at the tables.

Sources

Fast Food Source Document

Parent Requirements

overall requirement

Documenting Sources

Figure A.11 Requirement Extractor

text from the document into the definition of a requirement being created, the user can simply highlight the desired text in the document window and press the button corresponding the desired attribute. For example, the highlighted text in the document pane in Figure A.11 can be transferred to the paragraph title pane by pressing the Paragraph Title button. Of course, the text can also be moved by using the standard Macintosh **Copy** and **Paste** commands. However, this would require the user to enter two commands every time text needed to be moved.

In addition to being able to specify the values of attributes when the requirement is being created, the user can add targets to two relationships. The source which documents the requirement can be specified in the documenting sources pane. The requirements which incorporate the requirement being created can be specified in the parent requirements pane. The list panes along the right side of the window are selection panes which allow the user to add targets to these two relationships (see the **Add Parent** and **Add Source** commands).

Once the user has set the values for the attributes and relationships desired, the requirement can be created by simply pressing the button labeled Create Requirement. Then, if desired, the two clear buttons can be used to clear all values before the next requirement is defined. Though all necessary commands are accessible through the requirement extractor window itself, the commands are also duplicated in the Extractor Menu in case the user prefers to issue a command in this way.

5.1 Extractor Menu

The Extractor Menu contains commands to allow the user to define a requirement and then enter that requirement into the system database. These commands include: **Create Requirement, Add Parent, Remove Parent, Add Source, Remove Source,** and **Refresh Document.**

5.1.1 Create Requirement

The command **Create Requirement** allows the user to define a new element of type OriginatingRequirement with the attribute values specified in the attribute panes. The requirements in the parent requirements list are added to the relationship "incorporated by". The sources in the documenting sources list are added to the relationship "documented by". A requirement can also be created by clicking on the Create Requirement button in the requirement extractor window.

5.1.2 Add Parent

The command **Add Parent** allows the user to add the selected requirement in the requirement pane to the set of parent requirements. Double-clicking on a requirement in the requirement pane is a shortcut for this command.

5.1.3 Remove Parent

The command **Remove Parent** allows the user to remove the selected requirement in the parent requirements pane from the set of parent requirements. Double-clicking on a requirement in the parent requirements pane is a shortcut for this command.

5.1.4 Add Source

The command **Add Source** allows the user to add the selected source in the source pane to the set of documenting sources. Double-clicking on a source in the source pane is a shortcut for this command.

5.1.5 Remove Source

The command **Remove Source** allows the user to remove the selected source in the documenting sources pane from the set of documenting sources. Double-clicking on a source in the documenting sources pane is a shortcut for this command.

5.1.6 Refresh Document

The command **Refresh Document** allows the user to return the document displayed in the document pane to its original state as read from a disk file. Because the document can often be inadvertently altered by the user during the process of extracting requirements, the refresh command can be very useful.

6. Schema Extender

While the element editor, text view, hierarchy diagram, and requirement extractor are interfaces between the user and the contents of the system database, the schema extender allows the user to modify and extend the basic structure of the ERA database. Because the extender allows the user to change the database structure, the database must be empty when the extender is opened.

The schema extender consists of four list panes as shown in Figure A.12. In the

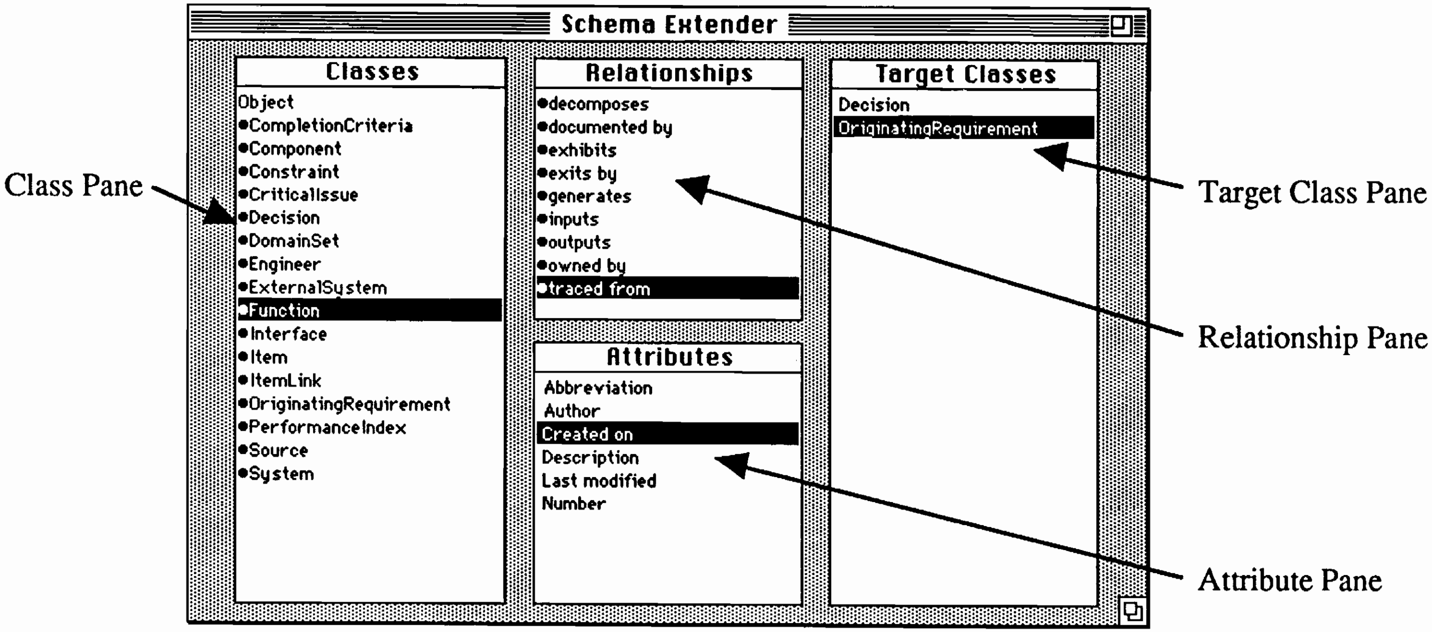


Figure A.12 Schema Extender

first list pane, the classes (e.g., Component, Function) are displayed. When the user selects the desired class, the relationships and attributes for the class are displayed. The relationship pane lists all possible relationships for the selected class, and the attribute pane lists all attributes (both those that are inherited and those defined for the selected class). When the user selects a relationship, all possible target classes are listed in the target classes pane. In Figure A.12, the relationship "traced from" is selected, and the possible target classes (Decision and OriginatingRequirement) are displayed.

If a name in the class or the relationship pane is preceded by a black circle (•), that class or relationship is consistent. A relationship is consistent if it has at least one target class. A class is consist if it has at least one relationship and if all relationships defined for the class are consistent. Obviously, if a relationship has no target classes, the relationship is not fully defined, and therefore the relationship is inconsistent. If a class has no relationships, the elements in that class cannot be related to any elements in the system. Because the existence of free-floating elements in a system defeats the purpose of complete system traceability, classes without relationships are defined to be inconsistent.

As shown in Figure A.12, the first class in the class list pane is the superclass Object. This class is an abstract class. Unlike the other classes in the system, no elements of type Object can be defined. The sole purpose for this class is to define the structure of the other classes which inherit the characteristics of the superclass. Therefore, the attributes of class Object are automatically inherited by all other classes. However, though classes inherit attributes, they do not inherit relationships. Therefore, relationships cannot be defined for the superclass.

Commands for the text view are contained in two menus. The Schema Menu contains commands to work with the entire schema. The Extender Menu contains commands to manipulate individual classes, attributes, relationships, and target classes.

6.1 Schema Menu

The Schema Menu contains all commands that deal with the entire schema as well as the command to close the schema extender. These commands include: **Import Schema**, **Export Schema**, **Return to Baseline Schema**, **Print Schema Report**, **Run Consistency Check**, and **Close Extender**.

6.1.1 Import Schema

The command **Import Schema** allows the user to load a schema file from disk into memory. The user is presented with a standard Macintosh dialog to select a text file to open. This command automatically deletes any schema extensions currently in the system. Therefore, the user is warned of this consequence before continuing.

6.1.2 Export Schema

The command **Export Schema** allows the user to write the schema structure currently in the system to a text file on disk. By exporting a schema, the user can create a relatively small, stable backup to the schema stored in the image file. Because the export includes only the schema extensions, it requires only a fraction of the disk space of an image file. Furthermore, since the export is a text file instead of a binary file, it is much less likely to become corrupted (since displacing a single bit in a text file will result only in a single character being changed whereas displacing a single bit in a binary file can corrupt all data in the file). Therefore, because the exported file is smaller and more stable than the original image file, the exported file is superior for storing and sharing a schema between multiple images.

6.1.3 Return to Baseline Schema

The command **Return to Baseline Schema** causes the image to erase all changes to the schema currently in the system and restores the schema to its baseline configuration. Because of the potential loss of schema data, the user is asked to confirm the choice to erase changes to the schema before any action is taken.

6.1.4 Print Schema Report

The command **Print Schema Report** allows the user to create a structured schema text report file on disk describing the current schema. The report can be opened via the **Open** command in the file menu and printed in order to obtain a hard copy. Furthermore, because the report is stored in a formatted text file, it can be read (and modified) by any word processor in order to be included in other reports.

6.1.5 Run Consistency Check

The command **Run Consistency Check** allows the user to perform a static check to determine if any inconsistencies exist in the current schema. If any of the following conditions is met, the schema is classified as inconsistent and appropriate messages are displayed in a consistency check window:

- A class has no relationships
- A relationship has no target classes.

If no inconsistencies are detected in the schema, a message is displayed in the consistency check window stating that the schema is consistent.

6.1.6 Close Extender

The command **Close Extender** invokes the schema consistency check to detect any inconsistencies in the current schema. If any inconsistencies are found, a consistency check window is opened, and messages documenting the inconsistencies are displayed. If no inconsistencies are encountered, the extender window is closed, the database is built, and the element editor window is opened.

6.2 Extender Menu

The Extender Menu contains the commands that allow the user to update the schema definition. These commands include: **Create Class, Rename Class, Remove Class, Add Attribute, Rename Attribute, Remove Attribute, Add Relationship, Remove Relationship, Add Target Class,** and **Remove Target Class.**

6.2.1 Create Class (Command + C)

The command **Create Class** allows the user to add a new subclass to the superclass Object. The user is prompted to enter a name for the new class. If the name does not conflict with a class already in the schema, the new class is created.

6.2.2 Rename Class

The command **Rename Class** allows the user to rename the selected class. If the selected class is part of the baseline schema, an error message is returned, because the user is only permitted to make additions to the baseline schema, not to change it. If the selected class is not part of the baseline schema, the user is prompted for a new name for the class. If the name does not conflict with a class already in the schema, the name of the selected class is changed throughout the schema (including all occasions in which the selected class is defined as a target class).

6.2.3 Remove Class

The command **Remove Class** allows the user to remove the selected class from the system as long as the selected class is not part of the baseline schema. Before the class is removed from the system, the user is asked to confirm the decision to remove the class. If the class is removed from the system, the class is automatically removed from the target class set of all relationships.

6.2.4 Add Attribute (Command + A)

The command **Add Attribute** allows the user to define a new attribute for the selected class. If the selected class is the superclass Object, the attribute is added to the superclass and is inherited by all other classes in the system. If the selected class is any other subclass, the attribute is added only to the selected class.

6.2.5 Rename Attribute

The command **Rename Attribute** allows the user to enter a new name for the selected attribute. If the selected attribute is inherited from the superclass Object, an error message is displayed. If the selected attribute is not inherited from the superclass Object and the attribute is not part of the baseline schema, the user is prompted for a new name for the attribute, and, if the new name does not conflict with the name of another attribute of the selected class, the selected attribute is renamed.

6.2.6 Remove Attribute

The command **Remove Attribute** allows the user to remove the selected attribute from the selected class. If the selected attribute is inherited from the superclass Object, an error message is displayed, because inherited attributes may only be removed from the superclass. Otherwise, if the attribute is not part of the baseline schema, the user is asked to confirm his choice, and the attribute is removed.

6.2.7 Add Relationship (Command + R)

The command **Add Relationship** allows the user to add a relationship to the selected class as long as the selected class is not the superclass Object. The user is prompted to enter a name for the relationship and its complement. If the relationship is not already defined for the class, the relationship is added to the class definition.

6.2.8 Remove Relationship

The command **Remove Relationship** allows the user to remove the selected relationship from the schema, if the selected relationship is not part of the baseline schema. All target classes are removed from the relationship. The complementary relationship is automatically removed from each target class if the selected class is the only target class for the selected relationship.

6.2.9 Add Target Class (Command + T)

The command **Add Target Class** allows the user to add multiple target classes to the selected relationship and automatically creates the complementary relationships for the target classes. A modal selection window consisting of a list of all classes in the schema and a set of five buttons (Add, Create, Remove, Done, and Cancel) as shown in Figure A.13 is opened. The behavior of the selection window is very similar to the behavior of the selection window to add targets to a relationship in an element.

When the selection window opens, all current target classes of the selected relationship are preceded by a white circle (°). As classes are added to the target class list, their names are preceded by a black circle (•). A class can be added to the target class list either by selecting the desired class and clicking on the Add button or by double-clicking on the desired class. If the user wishes to add a target class that does not yet exist, the user can create a new class by clicking the Create button. If a target class is inadvertently added to the target class list, it can be removed by selecting the target class and clicking Remove. However, pre-existing target classes cannot be removed in this manner. They must be removed via the command **Remove Target Class**.

Once all desired target classes have been added to the target class list, they can be added to the schema by clicking the Done button. Note that no changes are made to the schema until the Done button is pressed. Therefore, if the user wishes to close the window without changing the target classes of the selected relationship, the user can click on the Cancel button.

6.2.10 Remove Target Class

The command **Remove Target Class** allows the user to remove the selected target class from the selected relationship, if the selected target class is not part of the baseline schema. If the selected target class is part of the baseline schema, an error message is displayed.

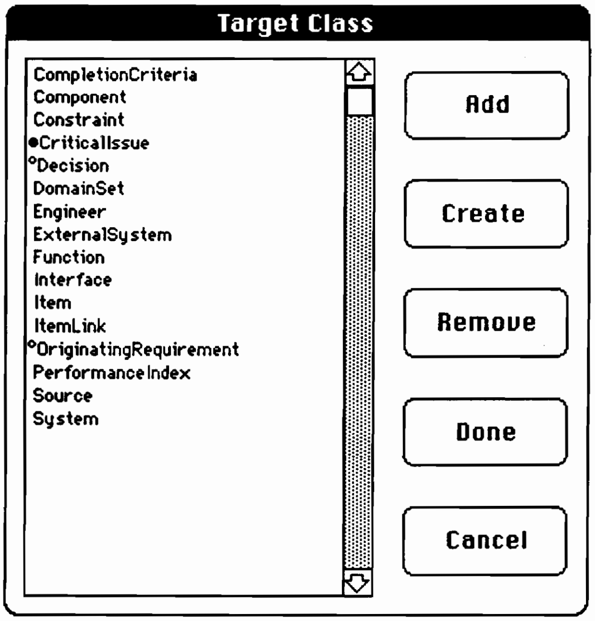


Figure A.13 Add Target Class Window

APPENDIX B: ENTITY-RELATIONSHIP DIAGRAMS FOR RATM BASELINE SCHEMA

The entire entity-relationship structure of the database is shown in matrix form in Figure B.1. However, though this matrix presents a complete view of the database structure, it can be very difficult to read. Therefore, entity-relationship diagrams are shown on a class-by-class basis in order to clearly show the structure of each class.

Relationships	allocated to (performs)	"	"	built in (built from)	"	"	carried by (carries)	connected to (connects to)	constrained by (constrains)	decomposes (decomposed by)	"	documented by (documents)	exhibits (exhibited by)	exits by (exit for)	generates (is generated by)	incorporates (incorporated by)	input to (inputs)	is basis of (is based on)	is contained by (contains)	output from (outputs)	owned by (owns)	referenced by (references)	traced from (traces to)	"	"
Target Classes	Component	ExternalSystem	System	Component	ExternalSystem	System	ItemLink	Interface	Constraint	Function	Item	Source	PerformanceIndex	CompletionCriteria	CriticalIssue	OriginatingRequirement	Function	PerformanceIndex	Interface	Function	Engineer	DomainSet	CriticalIssue	Decision	OriginatingRequirement
Classes																									
CompletionCriteria									●			●			●						●			●	●
Component				●	●	●		●	●			●	●		●						●			●	●
Constraint									●			●			●						●			●	●
CriticalIssue															●						●				
Decision												●			●						●		●		
DomainSet									●			●			●						●			●	●
ExternalSystem								●	●			●			●						●			●	●
Function	●	●	●						●	●		●	●	●	●						●			●	●
Interface									●			●			●						●			●	●
Item	●	●	●				●		●		●	●			●		●			●	●	●		●	●
ItemLink									●			●			●				●		●			●	●
OriginatingRequirement												●			●	●					●				
PerformanceIndex									●			●			●			●			●			●	●
Source															●						●				
System								●	●			●	●		●						●			●	●

Figure B.1 Baseline Schema Entity-Relationship Chart

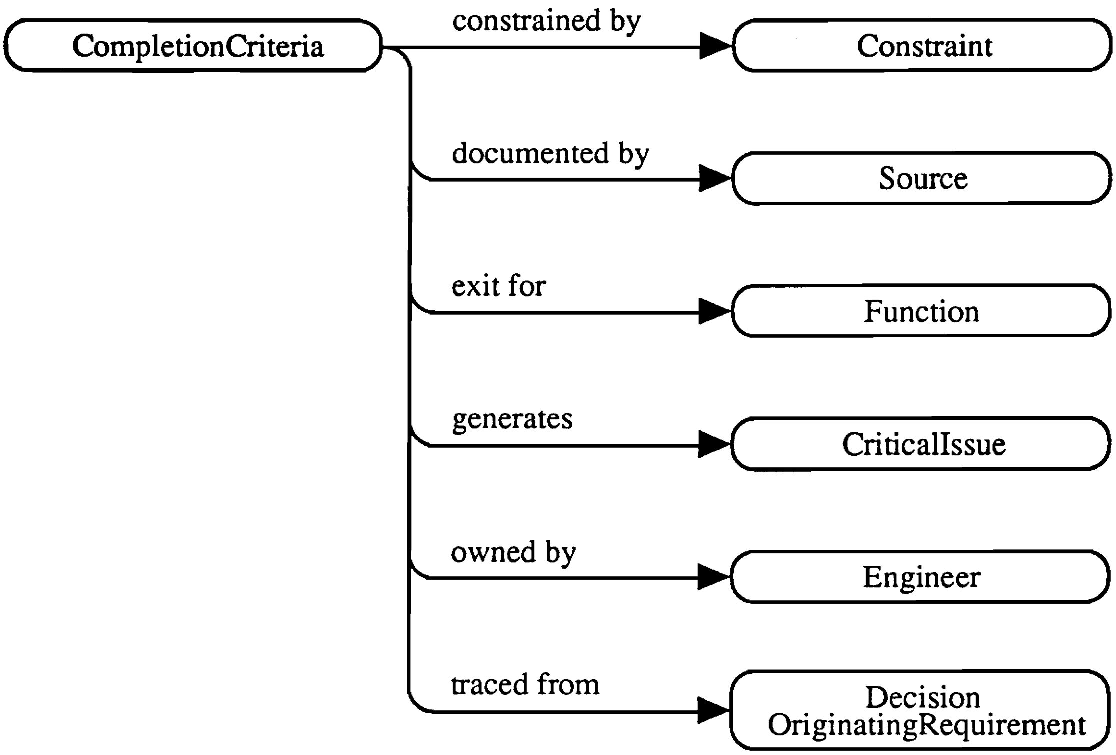


Figure B.2 Entity-Relationship Diagram for the CompletionCriteria Class

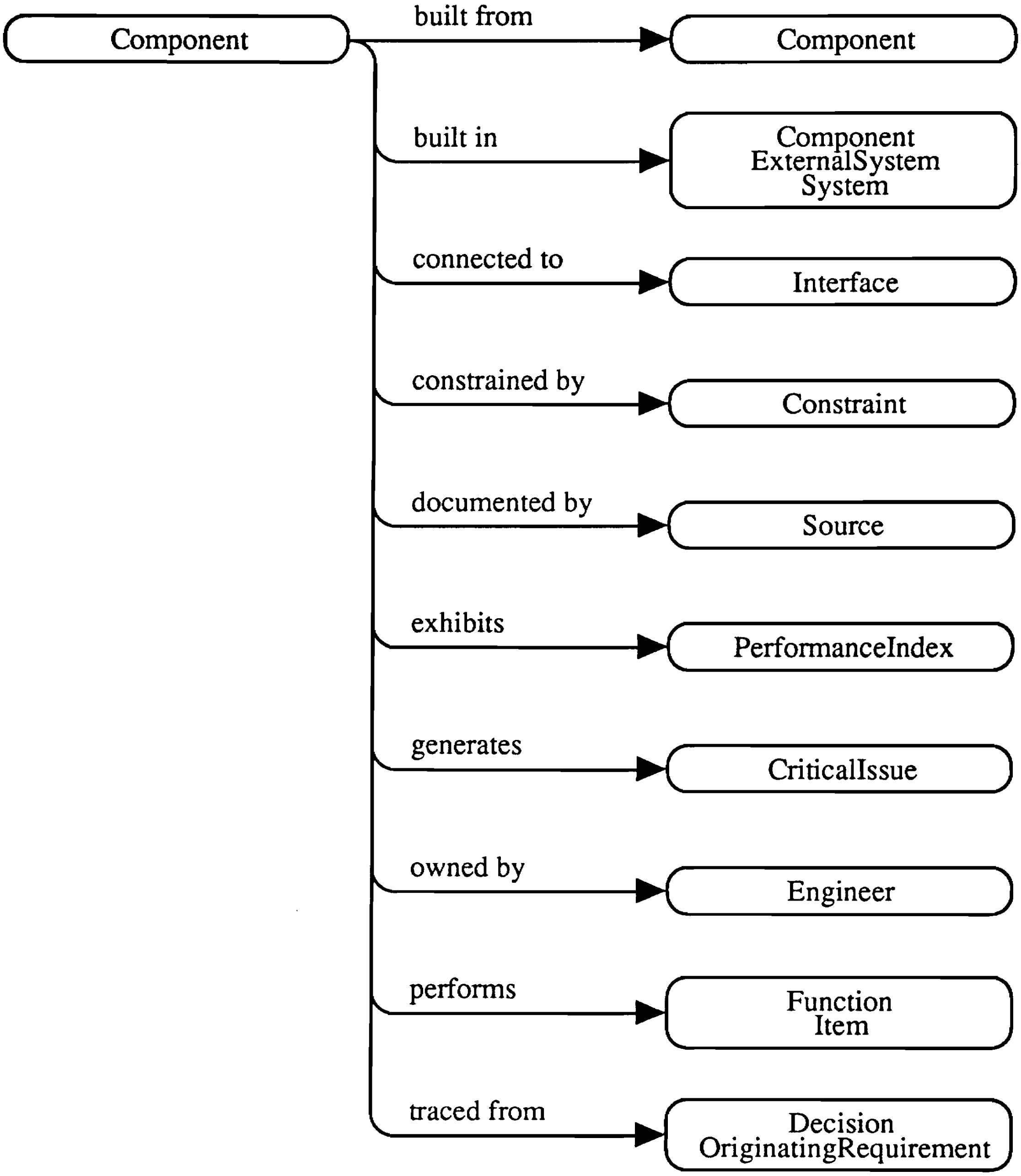


Figure B.3 Entity-Relationship Diagram for the Component Class

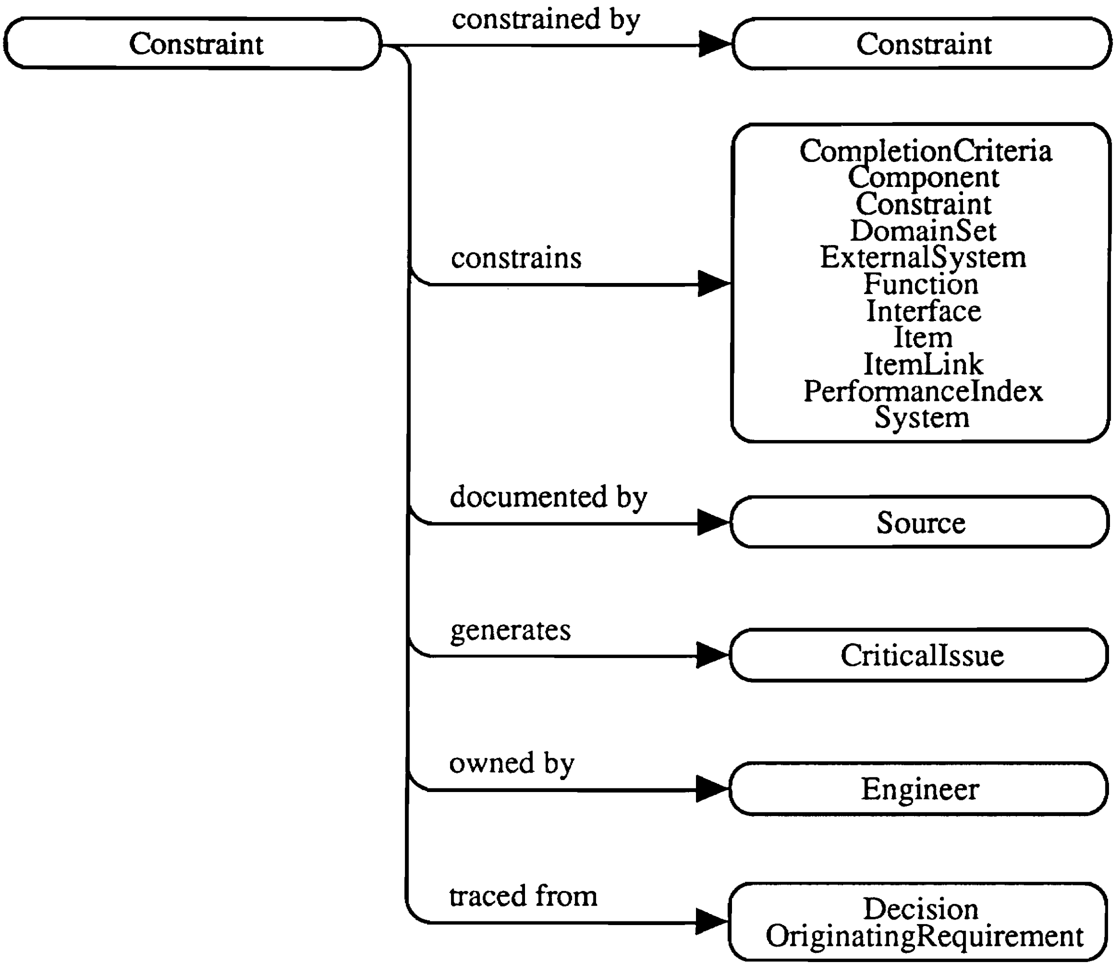


Figure B.4 Entity-Relationship Diagram for the Constraint Class

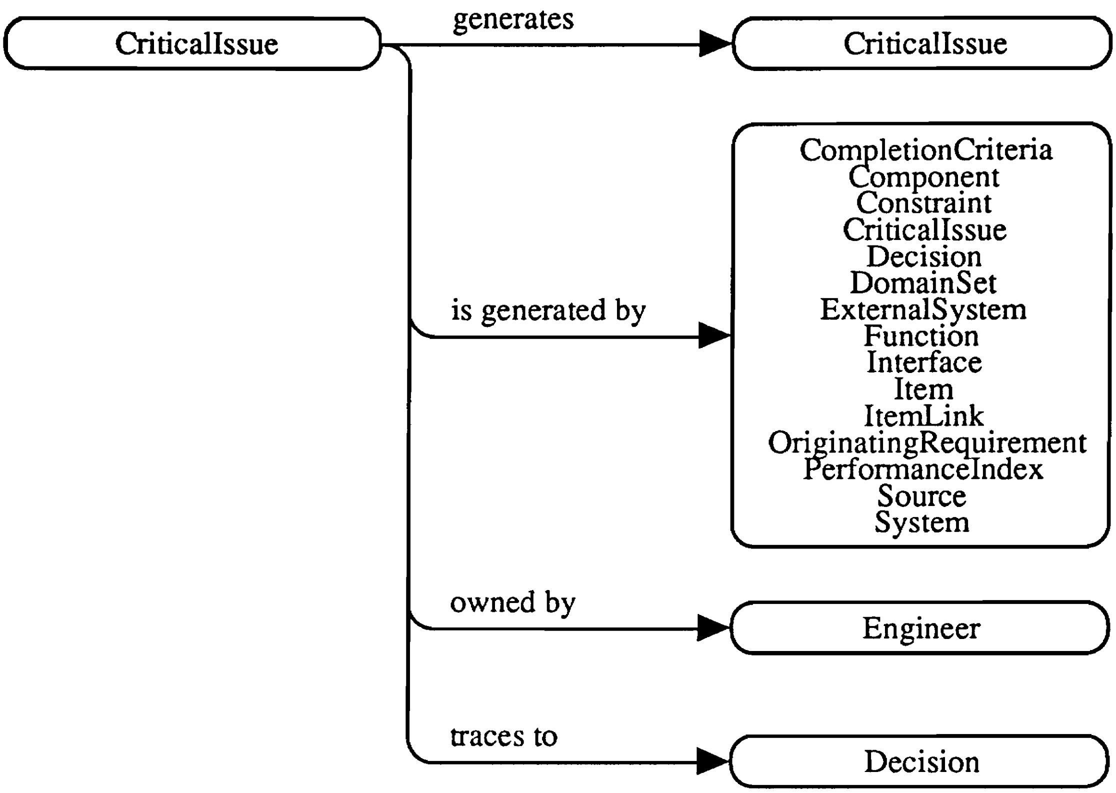


Figure B.5 Entity-Relationship Diagram for the CriticalIssue Class

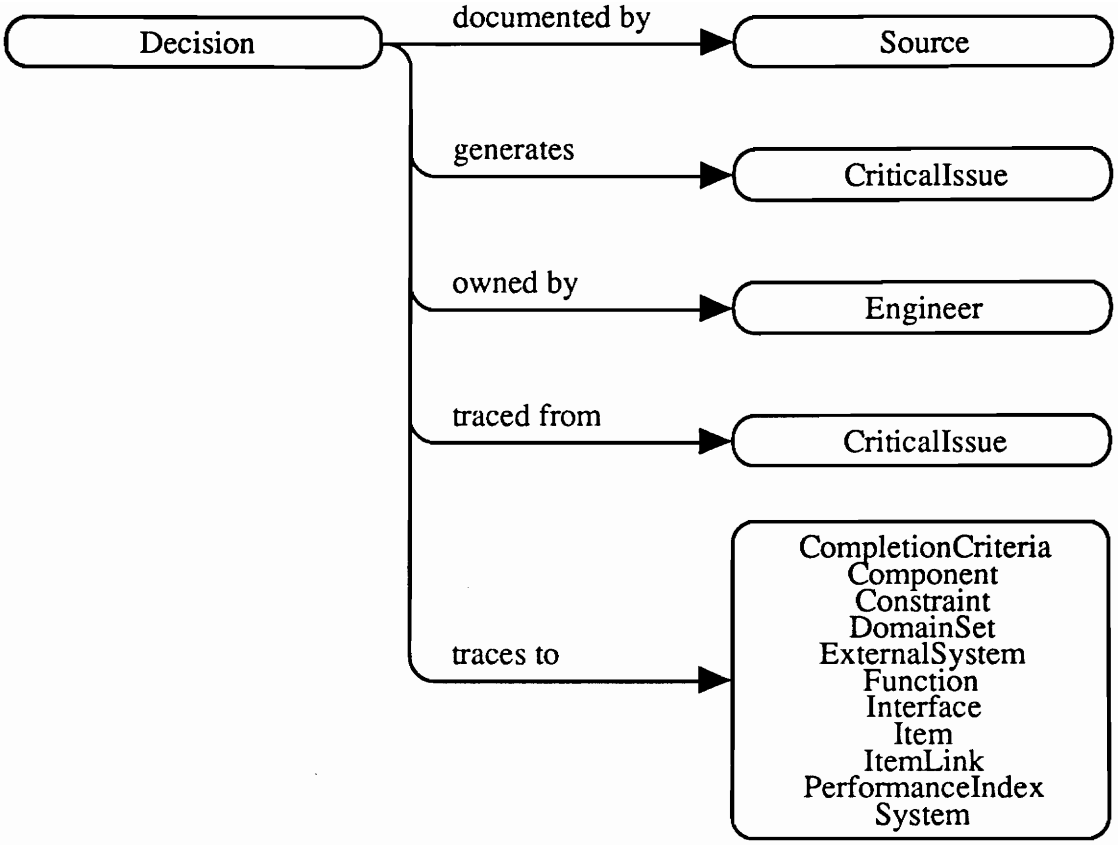


Figure B.6 Entity-Relationship Diagram for the Decision Class

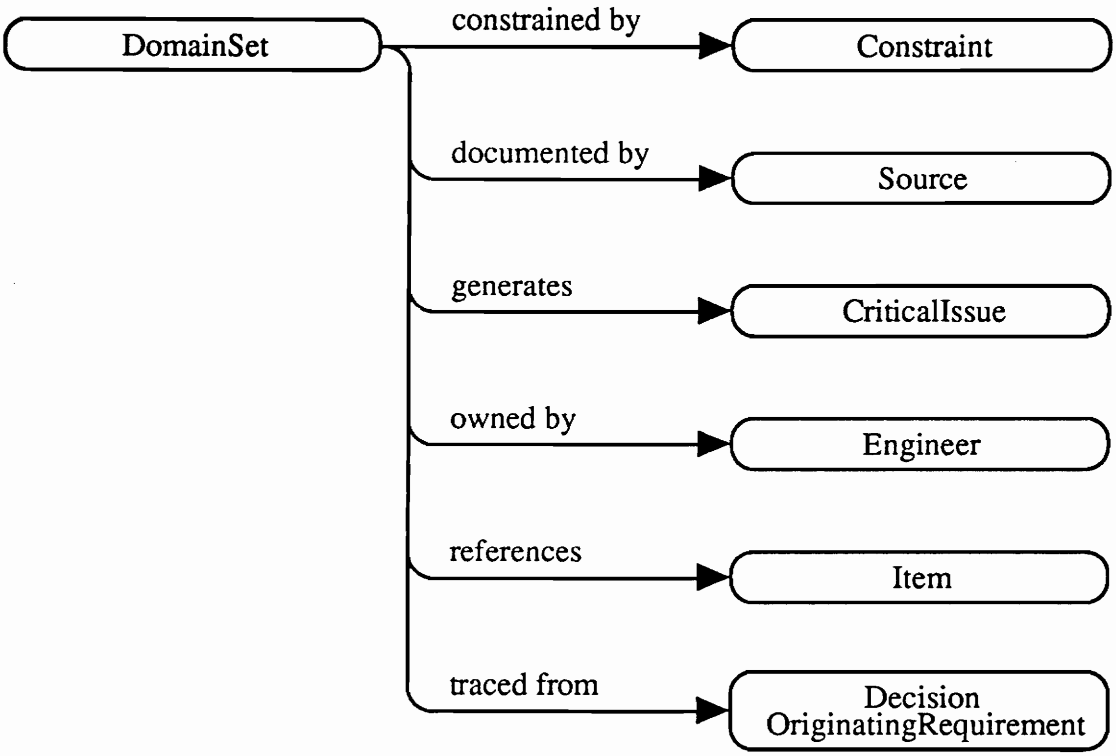


Figure B.7 Entity-Relationship Diagram for the DomainSet Class

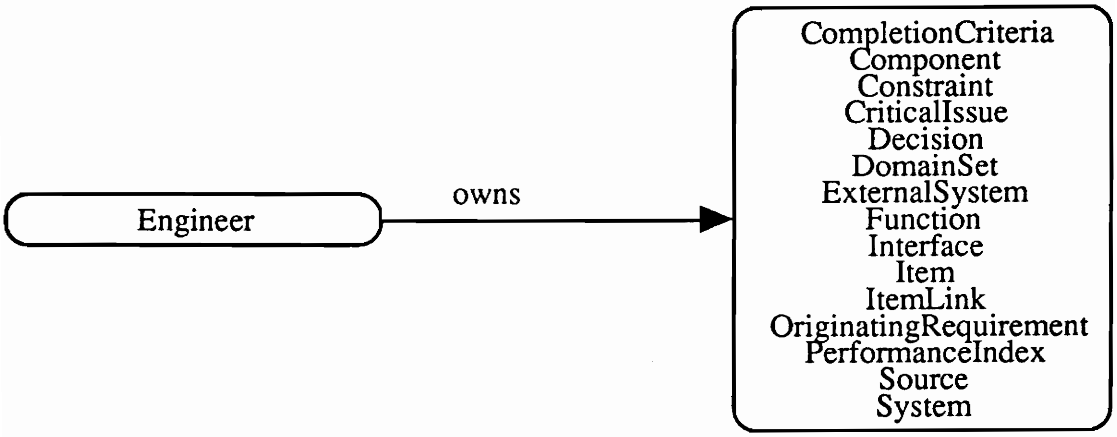


Figure B.8 Entity-Relationship Diagram for the Engineer Class

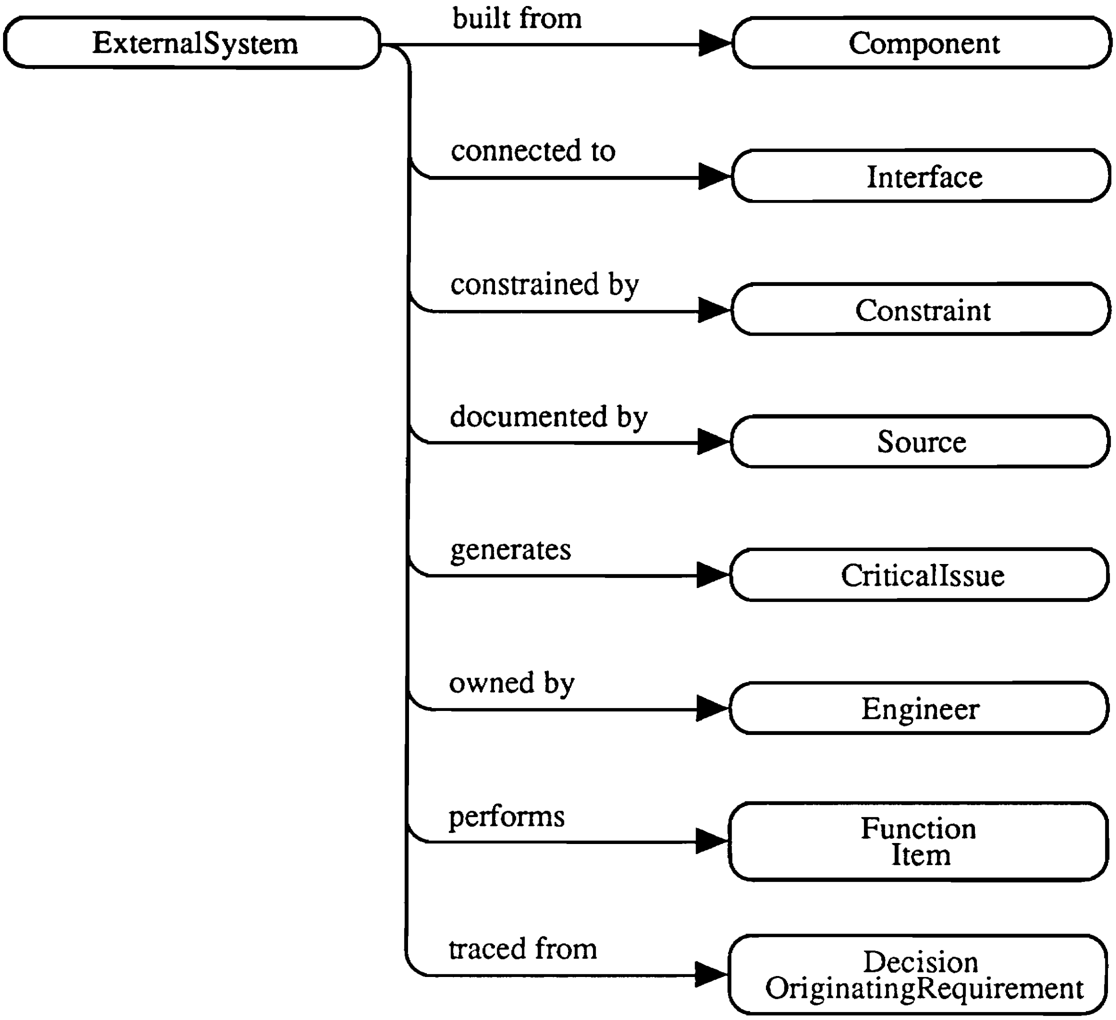


Figure B.9 Entity-Relationship Diagram for the ExternalSystem Class

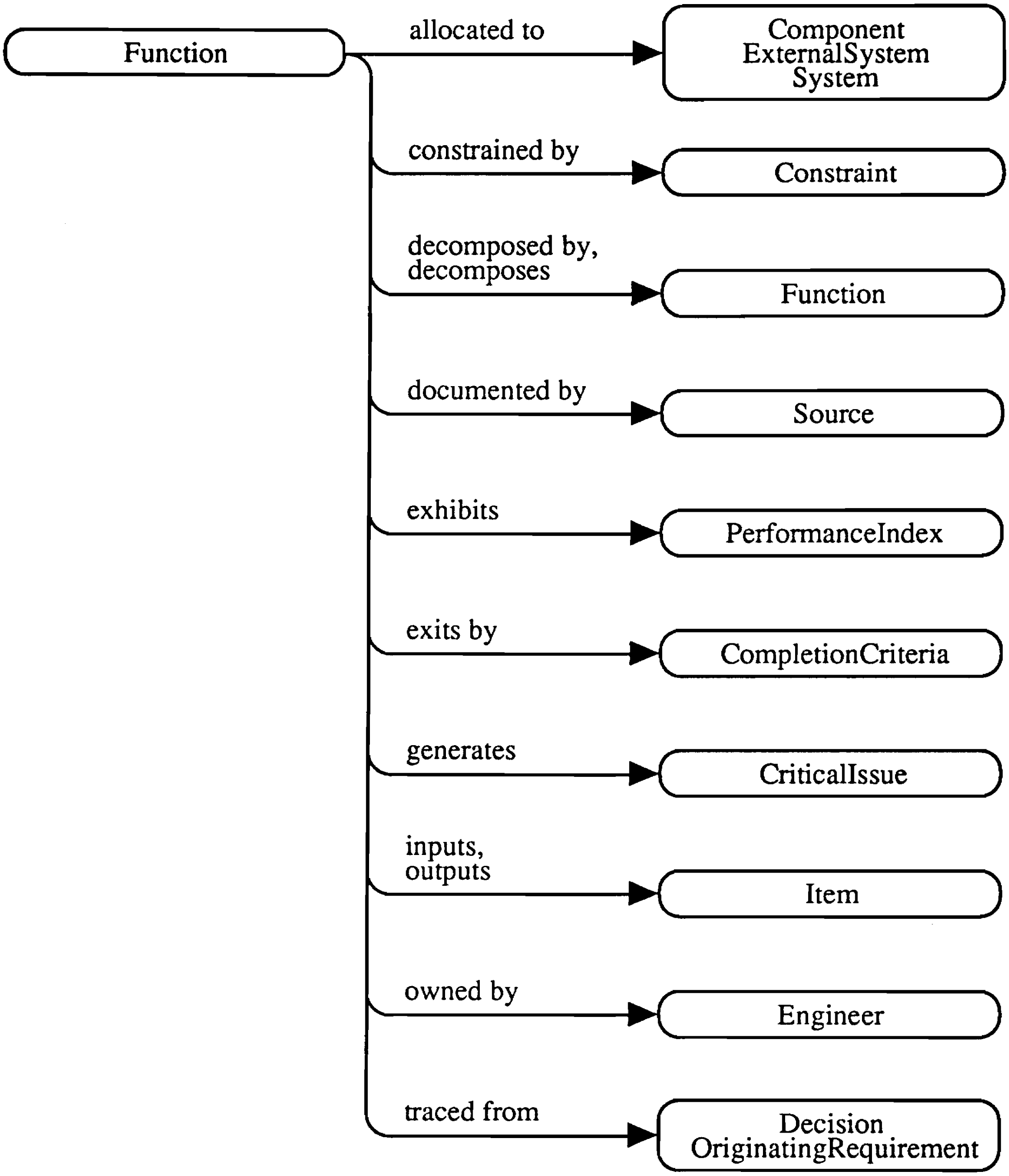


Figure B.10 Entity-Relationship Diagram for the Function Class

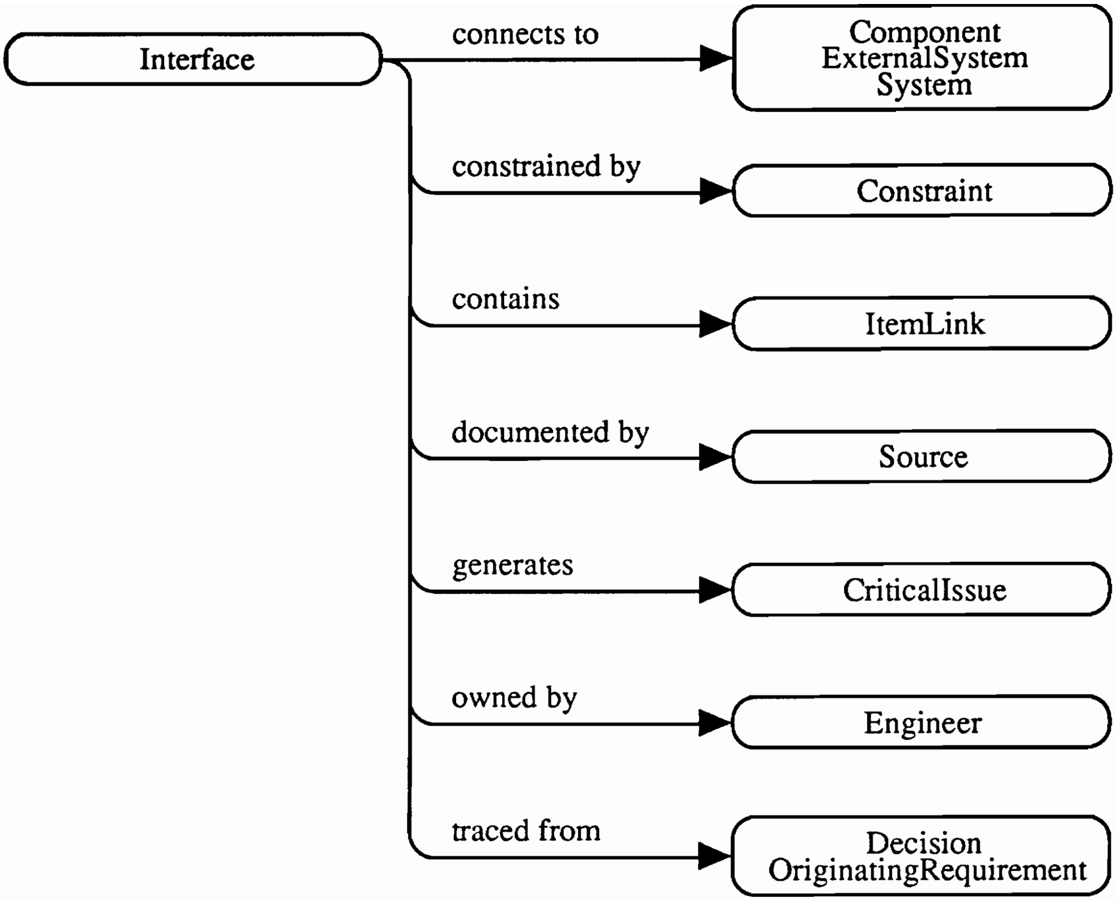


Figure B.11 Entity-Relationship Diagram for the Interface Class

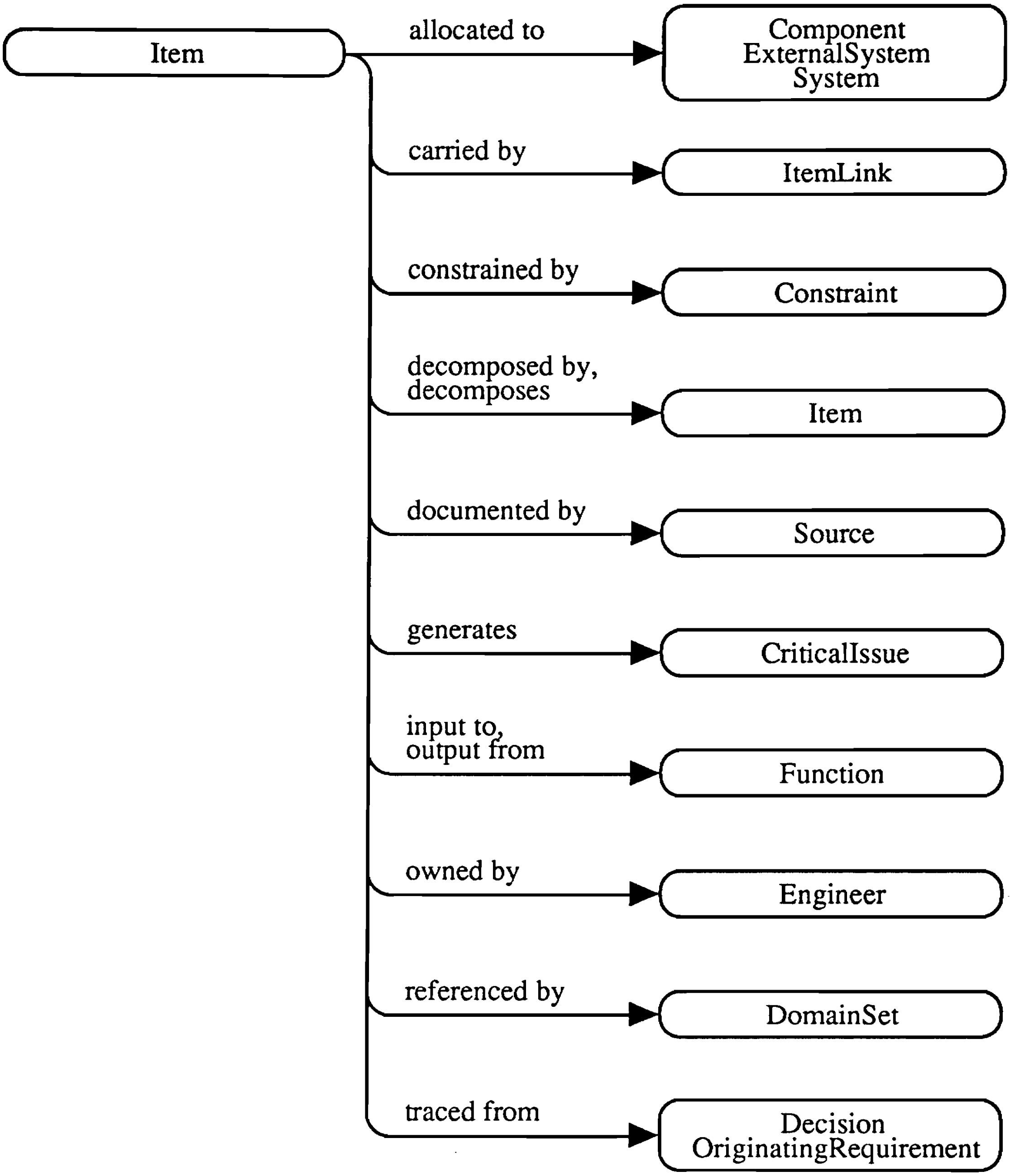


Figure B.12 Entity-Relationship Diagram for the Item Class

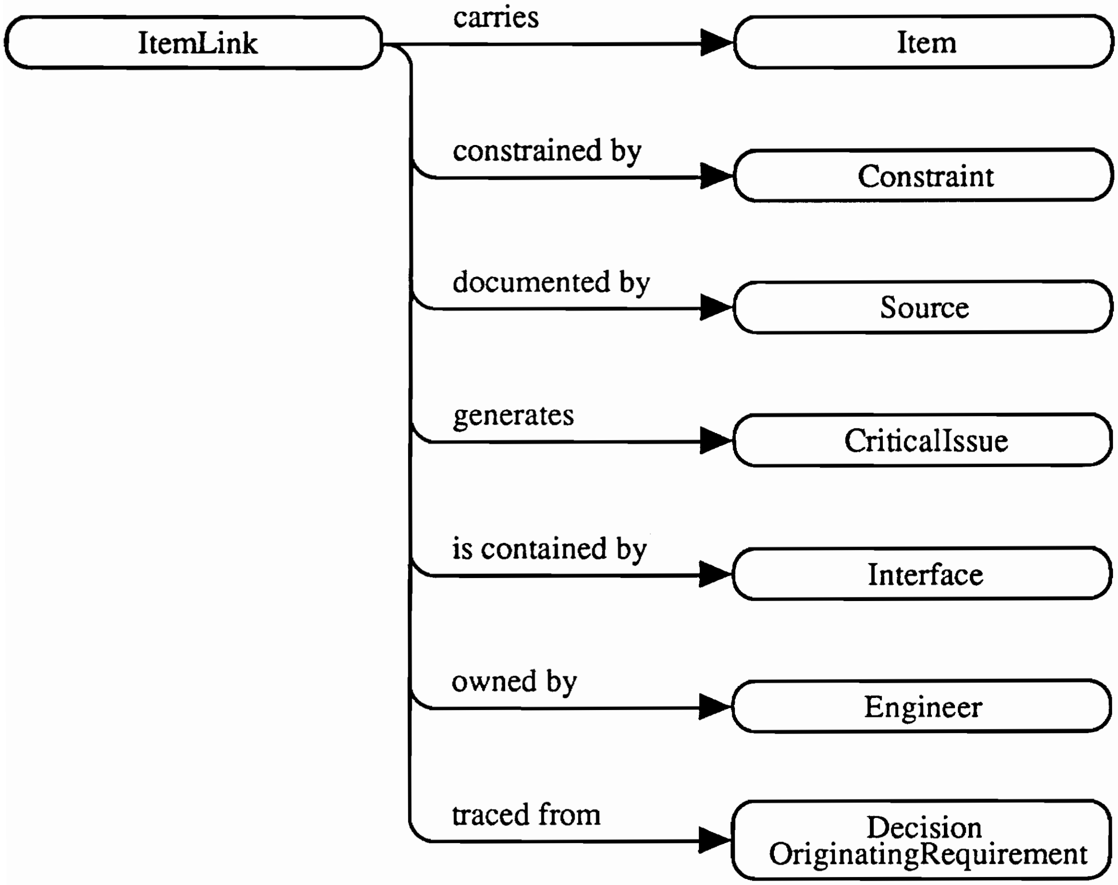


Figure B.13 Entity-Relationship Diagram for the ItemLink Class

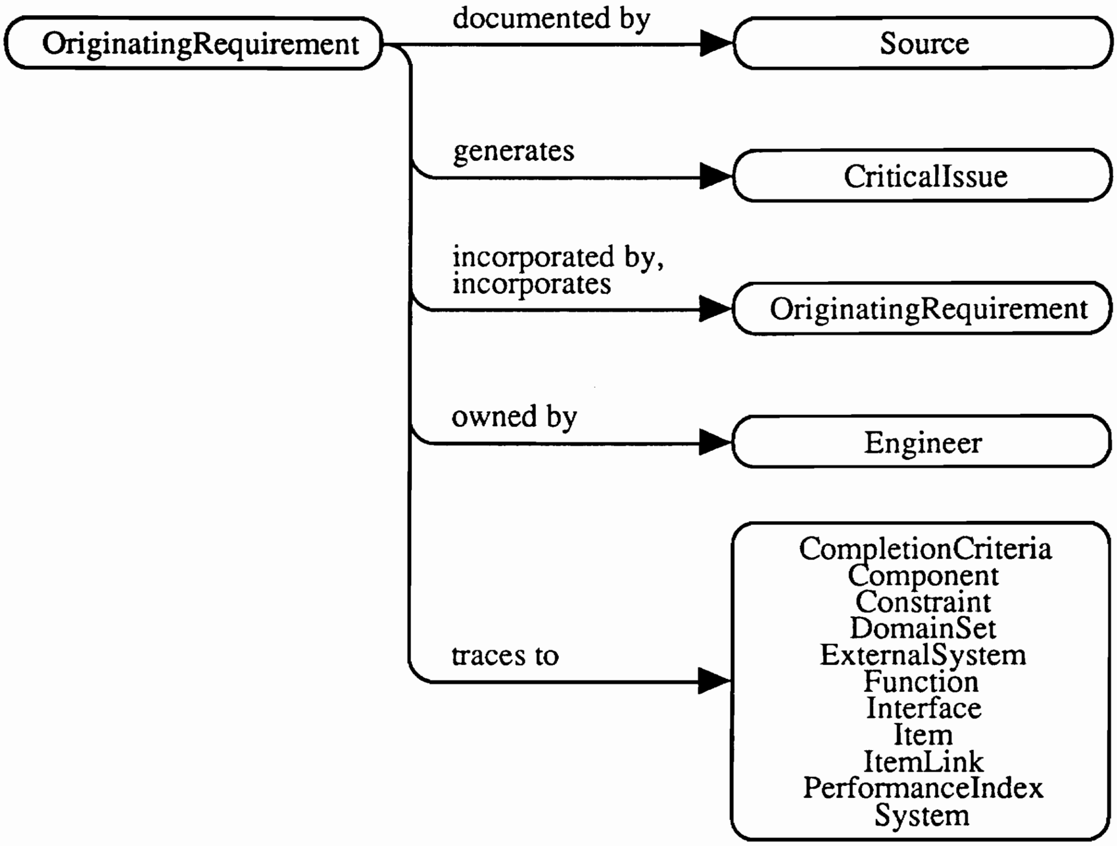


Figure B.14 Entity-Relationship Diagram for the OriginatingRequirement Class

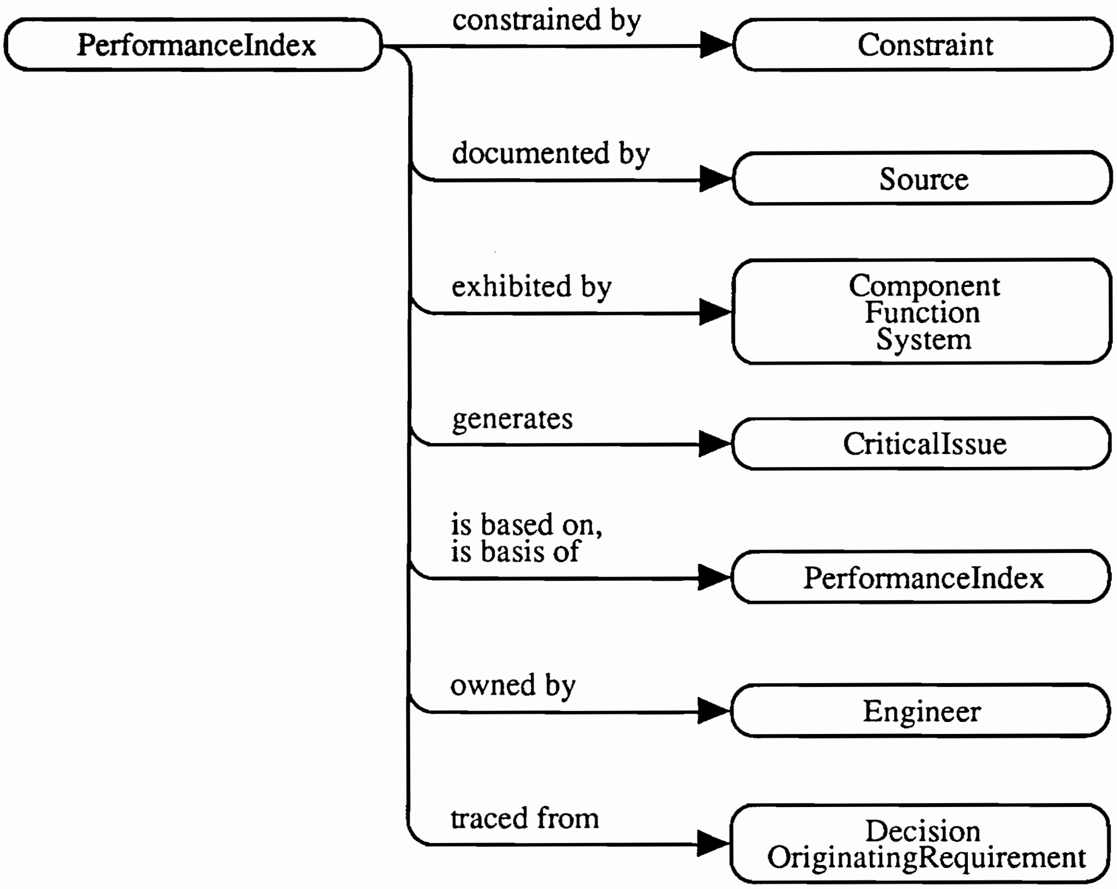


Figure B.15 Entity-Relationship Diagram for the PerformanceIndex Class

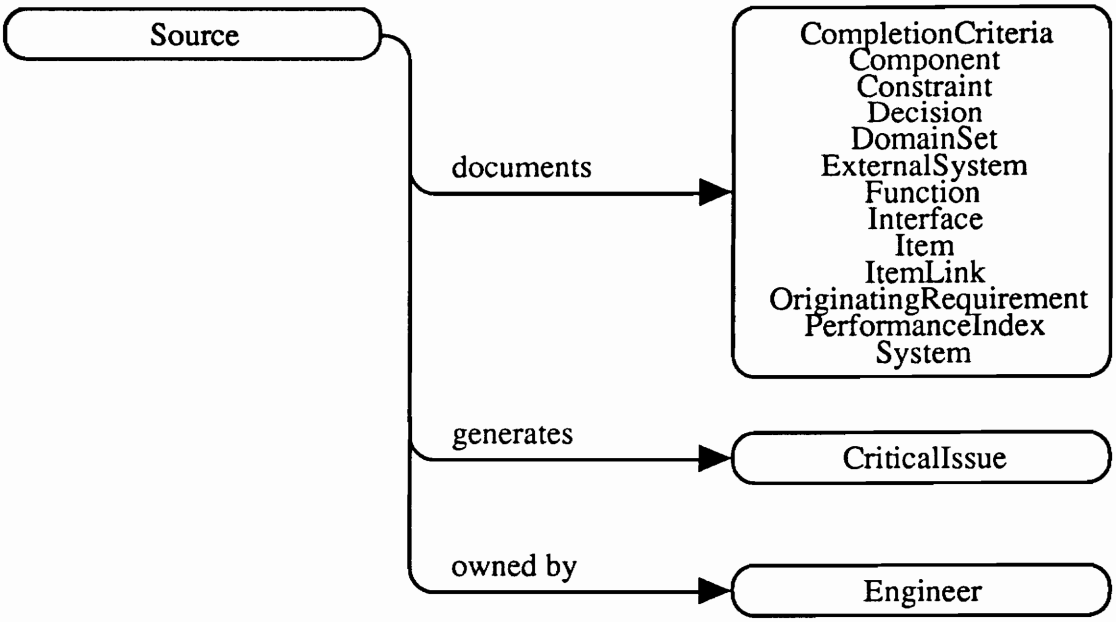


Figure B.16 Entity-Relationship Diagram for the Source Class

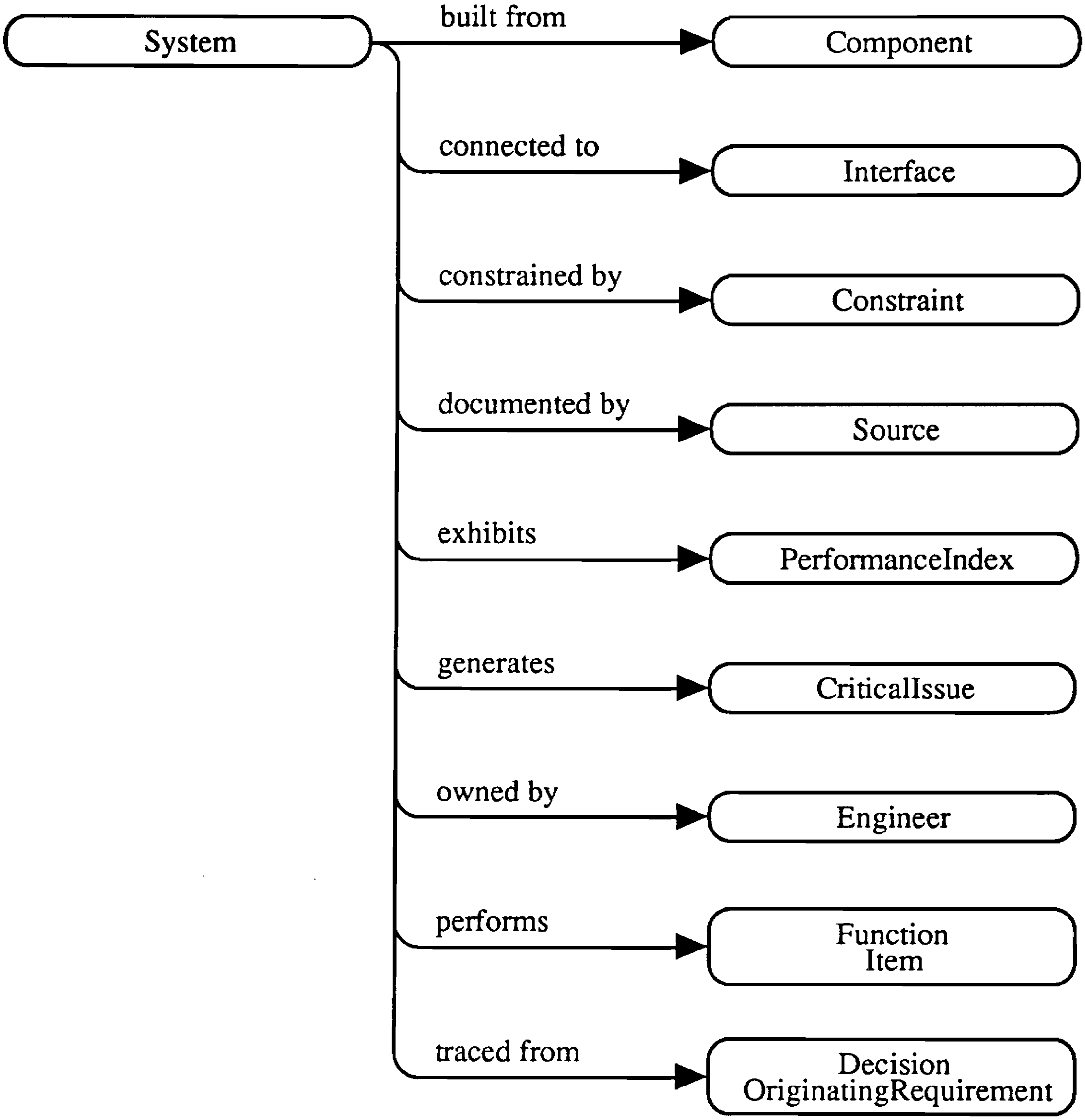


Figure B.17 Entity-Relationship Diagram for the System Class

VITA

David A. Long, born September 1, 1969, received the B.S. degree in engineering science and mechanics from Virginia Polytechnic Institute and State University in 1991. In 1991 he was named a Tau Beta Pi Fellow and attended graduate school in systems engineering at VPI&SU.

In the summer of 1989, he was employed by Omnitech Systems, Inc., Vienna, VA, where he worked with a state-of-the-art computer aided engineering system, RDD™ (Requirements Driven Development). During the summers of 1989, 1990, and 1991, he assisted in the development of system specifications and accompanying specification documentation. In 1992, he began to work full-time for Omnitech Systems where he developed the Requirements Analyzer and Traceability Manager engineering system.

David A Long

David A. Long