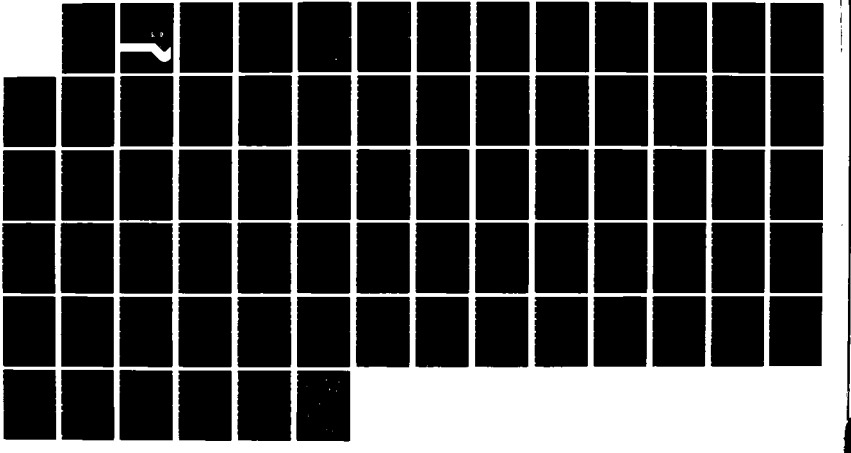
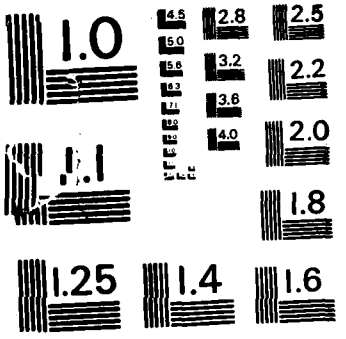
AD-A167 364 INTEGRATION OF HETEROGENEOUS BIBLIOGRAPHIC INFORMATION THROUGH DATA ABSTRACTIONS(U) LAWRENCE LIVERMORE NATIONAL LAB CA J O BREAZEAL JAN 86 UCRL-53710 1/1

UNCLASSIFIED W-7405-ENG-48 F/G 5/2 NL





UCRL-53710

Integration of Heterogeneous Bibliographic Information Through Data Abstractions

Juliette Ow Breazeal
(M.S. Thesis)

January 1986

Lawrence Livermore National Laboratory

DISCLAIMER

Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified/Unlimited	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UCRL-53710	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Lawrence Livermore National Laboratory 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) University of California Livermore, CA 94550	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Defense Technical Information Center 8b. OFFICE SYMBOL (If applicable) DTIC	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Cameron Station Alexandria, VA 22304-6145	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 65801S PROJECT NO. TASK NO. WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Integration of Heterogeneous Bibliographic Information Through Data Abstractions

12. PERSONAL AUTHOR(S)
Breazeal, J. O.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM _____ TO _____	860100	73

16. SUPPLEMENTARY NOTATION
M.S. Thesis

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
5	2		Intelligent Gateway, Post-Processing, Database Reformatting, Downloading
9	2		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In this study, heterogeneous bibliographic information resources from geographically distributed locations are integrated in an automated, unified, and controlled way by using abstract data types through the Message-Object Model as defined in Smalltalk-80. A unit of modularity call a "class" is developed that defines operations to process the data structures encapsulated in the class. The classes focus on processing bibliographic citations obtained from heterogeneous on-line bibliographic databases into a meta-form with the goal of developing information consistency to simplify further information analysis. Classes developed for the bibliographic citation application can speed program development because the data abstractions can be used in processing generic information such as dates regardless of the bibliographic database source. Prototype classes are developed to show the ease in encapsulating data structures and behaviors for the bibliographic citation application. Data abstractions provides a powerful integration technique that allow the designer to work with bibliographic citation objects without being encumbered with the details of implementation.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED/UNLIMITED
22a. NAME OF RESPONSIBLE INDIVIDUAL GLADYS A. COTTER	22b. TELEPHONE (Include Area Code) (202)274-5367 22c. OFFICE SYMBOL DTIC-EB

UCRL-53710
Distribution Category UC-32

Integration of Heterogeneous Bibliographic Information Through Data Abstractions

Juliette Ow Breazeal

(M.S. Thesis)

Manuscript date: January 1986

LAWRENCE LIVERMORE NATIONAL LABORATORY
University of California • Livermore, California • 94550

Available from: National Technical Information Service • U.S. Department of Commerce
5285 Port Royal Road • Springfield, VA 22161 • A03 • (Microfiche A01)

Integration of Heterogeneous Bibliographic Information Through Data Abstractions

By

JULIETTE OW BREAZEAL
A.B. (University of California, Los Angeles) 1960

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE
in

Computing Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Committee in Charge

Deposited in the University Library_____
Date Librarian

Abstract

In this study, heterogeneous bibliographic information resources from geographically distributed locations are integrated in an automated, unified and controlled way by using abstract data types through the Message-Object Model as defined in Smalltalk-80. A unit of modularity, ~~call~~ a "class" is developed that defines operations to process the data structures encapsulated in the class. The classes focus on processing bibliographic citations obtained from heterogeneous on-line bibliographic databases into a meta-form with the goal of developing information consistency to simplify further information analysis. Classes developed for the bibliographic citation application can speed program development because the data abstractions can be used in processing generic information such as dates regardless of the bibliographic database source. Prototype classes are developed to show the ease in encapsulating data structures and behaviors for the bibliographic citation application. Data abstractions provides a powerful integration technique that allow the designer to work with bibliographic citation objects without being encumbered with the details of implementation.

Keywords:

abstract data types, message-object model, class message, class methods, Smalltalk-80, Objective-C, information consistency, database consistency, database reformatting, database integration,

Acknowledgements

I wish to gratefully acknowledge the encouragement, dedication and guidance by Professor Meera Blattner, my thesis adviser, towards the success of this study. Both Dr. Hilary Burton and Professor Lawrence Kou have my gratitude for their help in reviewing this study and providing excellent comments and suggestions. My thanks are extended to the Technology Information System Group, the Computing Research Group, and the Electronics Engineering Research Group at the Lawrence Livermore National Laboratory for their support and their cooperation. Finally, I thank my husband, Norman and my children, William and Cynthia for their devotion and patience during this academic endeavor.

Contents

Page

Chapter 1:

Introduction

In recent years, the development of abstraction mechanisms in languages has focused on abstract data types to "manage complexity by emphasizing what is significant to the user and suppressing what is not"[Sha84]. This has lead to modern programming languages such as Smalltalk-80, Flavors, and Ada. Software methodologies have been developed to address engineering concerns in requirements, specification, design, implementation, correctness, and reliability to reduce cost during the software development and maintenance phases. The use of abstractions to logically reduce the complexity of the task is aided by modern language mechanisms in that they provide the language constructs to encapsulate a logical data type and the operations associated with it. The language constructs of "Classes" or "Flavors" help in the abstraction process. This project is based the use of abstractions to obtain data consistency in heterogeneous databases. Our specific implementation was applied to bibliographic information. Similar techniques may be applied to other types of databases, as described in Chapter 7.

Bibliographic citation databases from heterogeneous information resources are used widely in research and development work. These databases are often accessed to do a subject search or to prepare a bibliography. The citations contained in the bibliographic databases may be large in number and collected over a long period of time. This process was done manually before computers became readily available, and was tedious and error prone. Today computers are used widely for this task. Modern computer automated tools have been developed to assist in such bibliographic processing and are continually being enhanced[Gol85].

A research task may consist of accessing several bibliographic systems such as DIALOG, INSPEC, NASA/RECON, DOE/RECON, or DOD/DROLS.

The respective retrieved citation lists are down-loaded into a user file for post-processing analysis. Each database has its own form because of independent development programs and a lack of generally accepted standards. Hence post-processing analysis on a database citation file requires an individualized software processing package for each citation database. Sometimes the user's files are merged if software is available to translate the files into a common format.

To analyze data from the down-loaded and merged files requires data consistency. Hence a prototype has been developed to provide the tool to make heterogeneous bibliographic citation databases consistent. For example, the search for citations within a range of dates is encumbered by the problem that dates may be represented in different formats in different databases. Searches on author names are also a problem if different databases enter first, middle or last names in varying formats. The goal is to have one tool process the heterogeneous bibliographic citations into a standard form to provide the basis for convenient data analysis.

Significant improvements are made by conceptualizing the problem of data consistency by abstractions in terms of Smalltalk classes. Since the information types in citations are broadly similar, classes can be developed for each type of information such as "date" or "title". Careful specification of the classes can simplify the programmers task since interfaces will be defined, and data and their behaviors will be understood. Another improvement occurs when future enhancements are built on the classes already developed and serve to reduce the amount of new software needed.

This study shows the ease in developing prototype classes for integrating heterogeneous bibliographic citation databases and suggests the basis for the development of additional classes required for the complete application. The modularity of software, the inheritance by classes, the encapsulation of data structures and operations, and the use of dynamic binding reduce the task of the software designer and developer. Hence the Object-Message abstraction narrows the gap between the concepts and analysis of the problem and the notation used in the computer software to solve the problem.

In the following chapters, we discuss the background, motivation, and development of abstract data types via Smalltalk-80 classes to solve the problem of data consistency in heterogeneous bibliographic citation databases.

Chapter 2 discusses previous methods used. Chapter 3 gives the characteristics of the Message-Object Model. Chapter 4 discusses the physical hardware and software methods used to create the Objective-C classes for the prototype. Chapter 5 discusses the specifics of the prototype implementation, Chapter 6 discusses the results of the prototype implementation, and the last chapter discusses future directions.

Chapter 2:

Previous Methods for Processing Heterogeneous Bibliographic Information

This chapter gives some background information on bibliographic citation databases and discusses previous methods for processing the information.

2.1 Why Use Heterogeneous Bibliographic Information Resources?

Hall and Brown provide a statistical study of the on-line bibliographic databases that is the basis of this section[Hal83]. Online databases have been available since the 1960s but have mostly been in-house. Since 1972, there has been a rapid growth of publicly accessible databases.

Table I

Number of Bibliographic References Available Online

in millions

1968	1972	1976	1980	1982
1/4	3	20	58	77

The current rate of addition is 8.7 million references per year. With duplication accounted for, the estimate is 50 million singular references available for use and six million additions to the reference pool made per year.

Parallel to the four-fold increase from 1976 to 1982, the growth in on-line use is estimated to be six fold as seen in Table II.

Table II

Bibliographic Searches on Public Systems in U.S.A. and Canada

in millions

1975	1977	1979	1981
1	2	4	6

There are four particularly predominant database services. They are listed in Table III. Each supplier strives for uncommon databases in their service. Nearly 20 percent of the important databases are not available from the four services.

Table III

Unique and Common Databases available from major suppliers

Supplier	BRS	DIALOG	IRS	ORBIT
Unique	8	39	5	24
Common	28	56	27	28
Total Number	36	95	32	52
Total Percent	21	55	18	30

The vast repertoire of information makes the access to heterogeneous bibliographic information an important resource to a researcher. From Table III, we see that a password to DIALOG gives access to fifty-five percent of the databases. An additional password to ORBIT gives a total access to seventy percent of the databases.

Up to 1984, more than 2453 citation and numeric data files were available from 362 on-line information vendors[Cua84]. Scientific disciplines are continually adding to the published set of abstracts and citations. Most on-line bibliographic information is still obtained in printed form after an on-line search. The vast amount of information needs a tool with a unified view to extract significant scientific and technological intelligence.

2.2 Description of Bibliographic Citations

To understand the problems involved in heterogeneous bibliographic citations a simple MEDLINE citation is described as it is mounted on BRS. Only six fields were selectively down-loaded.

Sample Bibliographic Citation

[AU] Bowry-T-R. Oywang-J. Lumba-M.

[IN] Department of Human Pathology, Faculty of Medicine, University of Nairobi, Kenya.

[TI] HBV infection: prevalence of core antibody and other markers in urban based black school children in Kenya.

[SO] Ann-Trop-Paediatr. 1983 Dec. 3(4). P 197-200.

[LG] EN..

[IS] 0272-4939

The AU represents author, with hyphens separating initials. LG represents language and IS is the accession number for the citation in the particular database. IN represents the institutional affiliation of the author, TI is the title, and SO is the source. The same bibliographic citation from a different database source may be formatted in a completely different way. Inconsistency in the detail field hinders information analysis[Gol85].

2.3 Processing of Heterogeneous Bibliographic Information

With the appropriate administrative requirements fulfilled, a user can down-load bibliographic records from a variety of on-line services such as BRS or DIALOG. Typically, an off-line printing follows a search, and is arranged in reversed chronological order. The need for computer based editing tools is a natural consequence. Rather than obtaining the down-loaded information in stacks of printout, the bibliographic citations are down-loaded to a disk file so a computer can be used for automated processing of the information. We observe two problems that exist in local processing of the file. The file must be translated into a common form to handle the different database formats for data tags and to handle the inconsistencies in the detail information associated with each data tag.

Tools to develop data consistency are available in most modern database management systems. Information consistency within a specific bibliographic database may also be augmented by locally developed software and procedures. The database administrator can use software tools to constrain data entry to meet certain requirements. The user may be required to enter data strictly in integer format within a certain range of values or character format within a certain string length. Furthermore, the user may be required to enter strings that are pre-defined in a dictionary for that attribute, such as one of eight acceptable colors. We can see at this point that information may be entered correctly into a particular database in formats that are singularly defined by the local database administrator. However, there may be inconsistent formats among the heterogeneous bibliographic databases because of a lack of standards and autonomous database development and administration. For example, dates can be constrained in a local database to be either May 1, 1985 or 1 May 1985 format.

There may be additional differences in upper/lower cases, abbreviations, spaces, or punctuation. These inconsistencies hinder the automated processing of bibliographic citations in the down-loaded disk file. Hence we find in processing a search based on date ranges, software must be written to handle the date discrepancies, or the search will be incomplete. Author names also introduce problems because R. L. Smith, Richard L. Smith, and Richard Lee Smith are the names of the same author. If one desires a list of articles written by R'chard Lee Smith after a certain date, the tabulation would be inaccurate.

A recent study on popular 'front-end systems' available on the market for processing bibliographic citations shows that the user has a limited choice of features such as down-loading and file creation.(i.e., SciMate, InSearch, CONIT)[Bol84]. Software is not available to address the problem of data consistency among heterogeneous databases.

Goldstein and Prettyman have developed software to process down-loaded citations with the goal of incorporating a specified reference format into manuscripts. In their work they encounter the typical problems of processing heterogeneous bibliographic citations.

They propose transforming each citation into the following canonical format.

Field #	Data Element	Tag
1	TYPE	TY
2	DATABASE	DB
3	TITLE	TI
4	AUTHOR	AU
5	SOURCE	SO
6	INSTITUTION	IN
7	NO. & TYPE MTG	NO
8	MEETING TITLE	TM
9	VOLUME NO.	VL
10	ISSUE	IS
11	MONTH (JOURNAL)	MO
12	DAY (JOURNAL)	DY
13	YEAR (JOURNAL)	YR
14	MONTH (MEETING)	MM
15	DAY (MEETING)	DM
16	YEAR (MEETING)	YM
17	PAGES	PG
18	TOTAL PAGES	TP
19	PUBLISHER	PU
20	PUBL. CITY	PT
21	PUBL. STATE	PS
22	PUBL. COUNTRY	PC
23	PUBLICATION YR	PY
24	MTG.CITY	MT
25	MTG.STATE	MS
26	MTG.COUNTRY	MC
27	REPORT NO.	RN
28	RETRIEVAL NO.	RG
29	ISSN NO.	SN
30	PART NUMBER	PN
31	CODEN	CD
32	NOTES	NT
33	EDITOR TYPE	ED
34	AVAILABILITY	AV
35	COPYRIGHT YEAR	CY
36	PUBL.AUTHOR	AA

The process is divided into three stages.

* Pre-Processing
* Parsing
* Post-Processing

Steps for pre-processing records down-loaded from heterogeneous databases into separate local files are:

1. translate field labels in all files to a common set;
2. include fields for, and add database and retrieval system names to all records;
3. merge all records into one file;
4. reorder the records into a format that is optimized for further processing;
5. determine and add the type of publication;
6. standardize the format of the author's name.

The parsing stage is to separate the complex source field into discrete information. Further details are found in Chapter 3.

Post-processing is to further format the information for consistency in the end-product application program. The end-product could be a statistical analysis based on certain keywords or a bibliography for a publication.

The post-processing tasks are:

1. conversion for case consistency;
2. standardize journal titles;
3. correct inconsistencies in format;
4. expand abbreviated titles;
5. add missing data;
6. make linkages between articles and proceedings; chapter and citations.

The Goldstein and Prettyman work involves knowing the database source and then writing specific software for that bibliographic database source. Their proposal for a canonical form for bibliographic

citation databases is an attempt to develop standardization regardless of the bibliographic citation sources.

A significant amount of work has been done in the processing of heterogeneous bibliographic citation databases by the Technology Information System(TIS) of the Lawrence Livermore National Laboratory(LLNL). They have been working on technology transfer through computer networks located nationally and abroad since 1975 and have developed the Integrated Information System (IIS) that manages information and resources on the TIS system. IIS supports the down-loading and analysis of bibliographic citations from heterogeneous database services. A major goal is to provide the capability to extract scientific and technological intelligence from the information contained in these databases. To accomplish this, software has been developed to process bibliographic citations from the federal information centers of the Department of Energy (DOE), the Department of Defense(DOD), and the National Aeronautics and Space Administration(NASA) as well as the three major U. S. commercial services --- Lockheed-DIALOG, SDC-ORBIT, and BRS. [Bol84]

The Integrated Information System (IIS) software package is menu-driven and provides for the following bibliographic database options:

[TRANSLATE]	translates citations to a standard format
[MERGE]	combines translated files from different sources into one file
[STAT]	creates a statistical profile of citations
[ANALYZE]	analyzes bibliographical text
[REVIEW]	permits on-line evaluation of citations for relevancy.
[CONCORD]	creates indexes by author, subject, descriptors, etc.
[PERMUTE]	issues multi-term statistics of the text in selected data fields
[CROSS]	cross-correlates the contents of data fields
[PLOT]	shows the number of citations by year in a graph
[DISPLAY]	displays the contents of any file on the CRT screen

TRANSLATE, MERGE, DISPLAY and REVIEW do the pre-processing steps 1-4 mentioned by Goldstein and Prettyman. ANALYZE, CONCORD, PERMUTE, CROSS, PLOT, and STAT allow the user to produce some trend analysis from the bibliographic citations that have gone through the preprocessing steps.

Currently, the trend analysis is not entirely accurate since the detail information is not entirely consistent. A closer examination of the pre-processed files shows dates in the following form:

1. 1 May 1985
2. May 1, 1985
3. 1985.
4. 5/1/1985
5. 1985
6. 5/1/85
7. May 1985
8. May , 1985

The job of producing a file that is consistent is time-consuming and difficult; duplicate bibliographic citations are not easily detected. A particular citation usually contains only a subset from the set of data tags and different databases may enter certain detail information under different data tags. An example is the the following:

<DATABASE SOURCE> DIALOG NTIS FILE 6

<TITLE> Online Directory of Databases for Material Properties

<DATABASE SOURCE> DOE/recon

<TITLE(MONO)> Online directory of databases for material properties

The purpose of this project is to further extend the consistency of the detail information found in a merged file that is the result of down-loading heterogenous bibliographic citation databases. It is through the development of abstract data type Smalltalk-80 classes that similar types of information can be standardized, regardless of source. The standardization of dates and authors and titles include accounting for spaces, punctuation, capitalization, and ordering.

Chapter 3:

The Message-Object Model

We first establish the foundation for using abstractions in software development. Next, we discuss the motivation for using abstract data types via Smalltalk 80 classes to solve the data consistency problem in heterogeneous bibliographic citation databases.

3.1 Abstraction Mechanisms in Modern Programming

Recent work in programming methodology has led to the recognition of three kinds of abstractions: control, procedural and data. A large effort has been expended in developing a modern programming methodology so software is constructed that is easy to understand, modify, maintain, and is reliable. The quality of a program depends on the programming methodology used. The effective utilization of the methodology is strongly dependent on the programming language selected for the software development. Certain concepts in the methodology may be difficult to put into place if the language does not provide the constructs that make the process automatic. The language does influence the way a programmer thinks and formulates ideas. A good match of the methodology and the language enhances the likehood that the methodology will be followed. An example would be to attempt to introduce the concept of block structure using Fortran 66. A better choice would be Pascal because the language supports block structured constructs. While it is true that software can be written in Fortran to simulate the methodology, the job is unnecessarily enlarged for the software implementer[Lis74][Lis77].

3.1.1 Software Abstractions

What do we mean by software abstractions? We mean that the abstraction isolates the use from the implementation. That is to say, that the abstraction can be used without the knowledge of how the implementation was carried out, and the implementation can be done without the knowledge of how it is to be used[Lis77]. In the early 1950s, we see the application of abstractions in terms of assembly language rather than machine language in terms of octal numbers. Three letter acronyms were used instead of an octal number that represented the operator. Operands were designated by symbolic labels rather than absolute addresses in memory. Early languages supported built-in data types like integer and reals. One did not think in terms of binary bits in a computer word at a certain physical location in memory. Later type checking aided in appropriate default conversions when a real number was added to an integer. Hence, the programmer was relieved of low level detail. Procedural and control abstractions were dominant. Sorting procedures and square root functions could be specified without requiring knowledge of the implementation, and the implementation could be done without knowledge of how they were to be used. Later, control abstractions such as do-loops were made available so the concept of iteration was abstracted by the language construct. Abstractions were treated as a program organization technique. Programmers could define macros and define new data types required by a specific problem. We note that data structures such as stacks and linked lists were first treated systematically in 1968. The idea of studying and formalizing programming activity dates back to this time[Sha84].

What was recognized in early 1970 was that programs were difficult to understand and maintain. With the infamous gotos that spanned a large number of software lines indiscriminately, the term "spaghetti code" evolved and was a familiar occurrence among programmers. Locality was advocated in terms of if-then-else or do-while control constructs. For a while, extensible languages were promoted because they allowed the programmer to add new control constructs and data types to the base language in an attempt to add clarity to the program and make the programmer's tasks easier. This idea became unpopular since it was difficult to keep independent extensions compatible, to organize the definitions so related information were grouped together, and to find a technique to describe the extensions accurately.

The need for more accurate specifications was recognized since programmers typically relied on procedure headers and parameter lists with accompanying text to define the procedural abstraction. This specification technique depended on individual styles, and some were well written and accurate, while others were vague or out of date.

3.1.2 Structured Programming Methodology

The structured programming methodology was developed in the 1970s to address these problems: to make programs reliable, easy to understand, develop and maintain. It detailed phases in software development, specified tools needed to assist in the process, and established tests and criteria for program correctness. Program development was to evolve top-down using the idea of abstractions. First the statement of the problem was presented and then successive refinements were made until the problem was finally solved. The idea is to start with a high level abstraction and then progress by problem decomposition to recognizing subsidiary abstractions. This is where we find modern programming languages as CLU, Alphard, ADA, Concurrent Pascal, Euclid, Gypsy, Mesa and Modula being developed to support the structured programming methodology[Sha84].

3.1.3 Abstract Data Types

Procedural and control abstractions were available but the idea of abstract data types needed promotion. Through abstract data types, the idea of locality would hence be further extended, making programs easier to design, implement, and maintain. Specifications would be easier to write because of the encapsulation of the data structures. Data behaviors could be defined only within the abstract data type. The requirements of a language supporting data abstractions developed. Linguistic constructs were needed that implemented data abstractions as a unit in terms of data representations and operations on the data. The construct would provide a mechanism by which the language would limit access to the representation except by the operations defined. Smalltalk is such a language with abstract data types in terms of classes. CLU has clusters; Ada has packages; Flav has flavors.

A basic concept is that the operations defined for a class must include all operations needed in handling the data structure. Usually the operations include create, modify, and access operations. The desirability of classes is that the language takes care of all the interface specifications, the names for instantiations of the classes, the assignment, argument passing and type correctness.

Essential to abstract data types is the primitive library that is provided with the compiler. Here typical abstract data types as arrays, AVL trees, bags, and dictionaries are provided from which the programmer can develop new abstract data types particular to the application. Inheritance is important in that new abstract data types that are defined are based on the properties defined in a primitive abstract data type. As a matter of fact the abstract data types are usually arranged in a hierarchical tree so that an abstract data type inherits all the properties defined between it and the root of the tree.

Abstract data types are the means by which the human can transform problem-domain concepts into the computer-domain model. In other words, the separation of specification and implementation is the desired result. The goal is that program correctness at the abstract level can be ascertained before the implementation. The phrases "abstract data types" and "object-oriented programming have been used in various contexts, from Simula and its derivatives such as Ada to powerful data description languages used in knowledge representation. The meaning we apply is in the Smalltalk-80 context.[Cox84]

3.2 Object-Oriented Programming

Object-Oriented Programming replaces the operator-operand concepts that are used in the traditional computer-domain model. The idea is to introduce a coordination tool that supports change, reusability, and enhancements. The goal is to transfer work from the human to the machine and to enhance consistency from the human viewpoint.

Two major concepts of Object-Oriented programming are encapsulation and inheritance. Encapsulation is an aid in using the system and isolates the objects from the environment except through a carefully controlled interface. Inheritance is a aid to building the system. New classes are defined by first inheriting the data and behaviors of older generic classes, then specifying only how the new ones differs. The idea is to define the data abstractions so the programming task is made minimal.

Now we will define some terms used in a Message-Object programming language such as Smalltalk-80. The terms object, message, class, instance, and method are all defined in terms of each other. We will relate the terms to the Objective-C compiler that is a derivative of Smalltalk-80, and will clarify them by examples in Chapter 5.

3.2.1 Objects

Objects are virtual(computer-based) machines. They have some data (private part), a set of operations(shared part), and a run-time mechanism for selecting operations on the data that are activated by a message sent to the object. They exhibit one of their behaviors when they receive a message.

3.2.2 Messages

Messages are sent to objects and are requests to obtain a desired result. The message contains a predefined operation(method) to be done on the data structure and are serviced one at a time by the object. Objects representing numbers have arithmetic operations; objects representing data structures as AVL trees create an empty tree, add, delete, modify, or count elements.

3.2.3 Classes

A class represents a description of a group of similar objects. A class is the abstract data type and an object is an instantiation of it. For example the class rectangle deals with the generic group of rectangles, but an instance of class rectangle will have specific dimensions for length and width. Binding is done at run-time so there is no static type checking at compile time. An example would be the class Array in Objective-C. The subclasses BytArray, IdArray, and IntArray inherit properties from class Array. Hence an operation as printOn defined in class Array will work on any of the three subclasses mentioned, although the data representations differ in terms of byte, Id, or integer. Also a new subclass defined later will also be handled correctly, and class Array does not have to be revised to make considerations for the new subclass data type. This is how reusability in data abstractions becomes a major asset in software development.

3.2.4 Methods

A method is a description of how to do an operation and is specific to the class in which it has been defined. It resembles procedures and could use class variables as parameters. Methods are written in a high level language like Smalltalk-80, Lisp, or C. The set of methods should include all the operations needed to work with the encapsulated data, either via inheritance or definition within the class.

3.3. Benefits of Object-Oriented Software

One basic caveat of object-oriented software is the concept of reliable reusable code. As a matter of fact the classes are called IC's from the engineering concept of integrated circuits. To start with, one uses a set of basic classes that form the root of the inheritance tree that can be systematically augmented by defining new classes.

To further understand the problem we are addressing, let us look at the Goldstein and Prettyman[Gol85] analysis of bibliographic sources from four different bibliographic citation databases: MEDLINE, INSPEC, ISIC, and COMPENDEX.

[MEDLINE] Ann-Trop-Paediatr. 1983 Dec. 17-18. 3(4). P 197-200.

[INSPEC] LASER FOCUS (USA). VOL.19, NO.8. 61-6.

[ISIC] COMPUTER 9(3):11-12

[COMPENDEX]

a) Electronics v 56 n 7 Apr 7 1983 p 155-157.

b) IEEE Trans Magn v Mag-14 n 5 Sep 1978, INTERMAG (Int Magn) Conf, Florence, Italy, May 9-12 1978 p 964-965.

The parsing of the citation source is a major task in arriving at the information in the canonical form suggested. It cannot be automated fully, and is iterative due to inconsistency in the data, addition of new words to the authority dictionaries, and new valid acronyms, entries and words.

Goldstein and Prettyman give an accompanying parsing structure for each of the above citation sources.

[MEDLINE] [title].*[year]*[month].*[day(s)].*[vol]([issue]).*P*[pages].

[INSPEC] [title]([country]).*VOL.[volume],*NO.[number].*[pages].

[ISIC] [title]*[volume]([issue]):[pages]

[COMPENDEX]

a) [title]*v*[volume]*n*[issue]*[month]*[day(s)]*[year]*p*[pages]

b) [title]*v*[volume]*n*[issue]*[month]*[days]*[year],*
[conf. name],*[city],*[country],*[month]*[day(s)]*
[year]*p*[pages].

One observes there are classes that are common to the different sources. As a matter of fact, the tasks involved in processing for data consistency of title, volume, and date, are similar regardless of the database origin or the citation source. There may be variations in case, punctuation, abbreviations, and/or format. We see date specified as Sep 1978 or May 9-12 1978 in the COMPENDEX sources. The goal of this project is to develop some prototype classes that augment the set of generic classes to provide the abstract data types needed to produce data consistency in citations from heterogeneous bibliographic databases.

Chapter 4:

Prototype Development Environment

This chapter describes the physical hardware and software methods used to implement the prototype classes to process heterogeneous bibliographic citation databases into a consistent form.

4.1 Computer System

The work was started on the LLNL Engineering Research Division (ERD) VAX 11/780 using the VMS operating system since it was the only installation with the Objective-C compiler at LLNL at the time. The parser development using the Unix tools LEX and YACC was done on the Tektronix 6205 workstation. The parser modules were sent over the network to the VAX to be compiled by Objective-C along with the prototype class modules to minimize use of the resources on the VAX. With limited system resources on the ERD VAX, the work was later completed on the LLNL Technology Information System (TIS), which meanwhile acquired the Objective-C compiler. Their VAX 11/780 uses the UNIX operating system BSD 4.2; certain VMS program lines needed for compatibility with Objective-C were removed. In general the environment was simpler for development work since the VMS port for the Objective-C was still in progress whereas the port for Unix BSD 4.2 was complete.

4.2 Software Development Tools

The Objective-C compiler from Productivity Products International in conjunction with the C compiler was used to implement the Object/Message model prototype for bibliographic citation databases. The Unix tools Lex and Yacc were used to develop the parser generator, and the tool Make aided in software development. [PPI85]

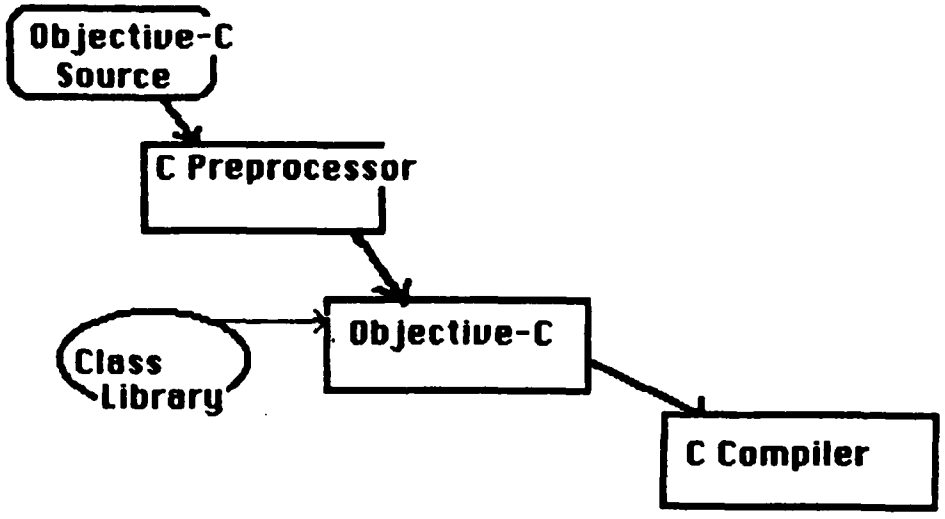
4.2.1 Objective-C Compiler

The Objective-C compiler is based on the Smalltalk-80 Message/Object Model. The syntax for developing classes in Objective-C resembles the Smalltalk-80 language but differs significantly in that the class methods are defined using the C-language. The Objective-C compiler is a preprocessor that produces C source that is then compiled. The preprocessor produces Class and Phylum files that are information repositories and form the basis for inheritance and encapsulation for the classes.

Smalltalk-80 is the result of 14 years of research and development by the Software Concepts Group at Xerox PARC. It is based on a software environment contained entirely within a workstation with special hardware to improve performance by orders of magnitude. The Smalltalk-80 environment solely uses the Smalltalk-80 language and provides the software person with a repertoire of basic classes. The environment includes utilities usually provided by the computer operating system, such as the text editor, compiler, and debugger. The environment makes extensive use of graphics windows, pull down menus, and a pointing device so the user can work on several views of his work in progress. To change text under software development, the user points at the line, edits it, issues the compile command, removes syntax errors, tests the software, and then compiles and links the new software into the system. All this is done without changing "modes" for editing, compiling, filing or executing.

The Objective-C compiler is different in that it is one of the many tools the software developer can add to the utilities offered by the operating system. It is available in the VAX VMS operating system environment as well as computer systems with the Unix BSD 4.2 operating system. It is is a preprocessor to the C compiler and adds the basic Smalltalk-80 concepts of classes, objects, messages, encapsulation and inheritance. Objective-C is an object oriented programming language layered on top of C and allows one to use it in addition to the existing software and hardware.

Diagram of Compilation Units[PPI85]



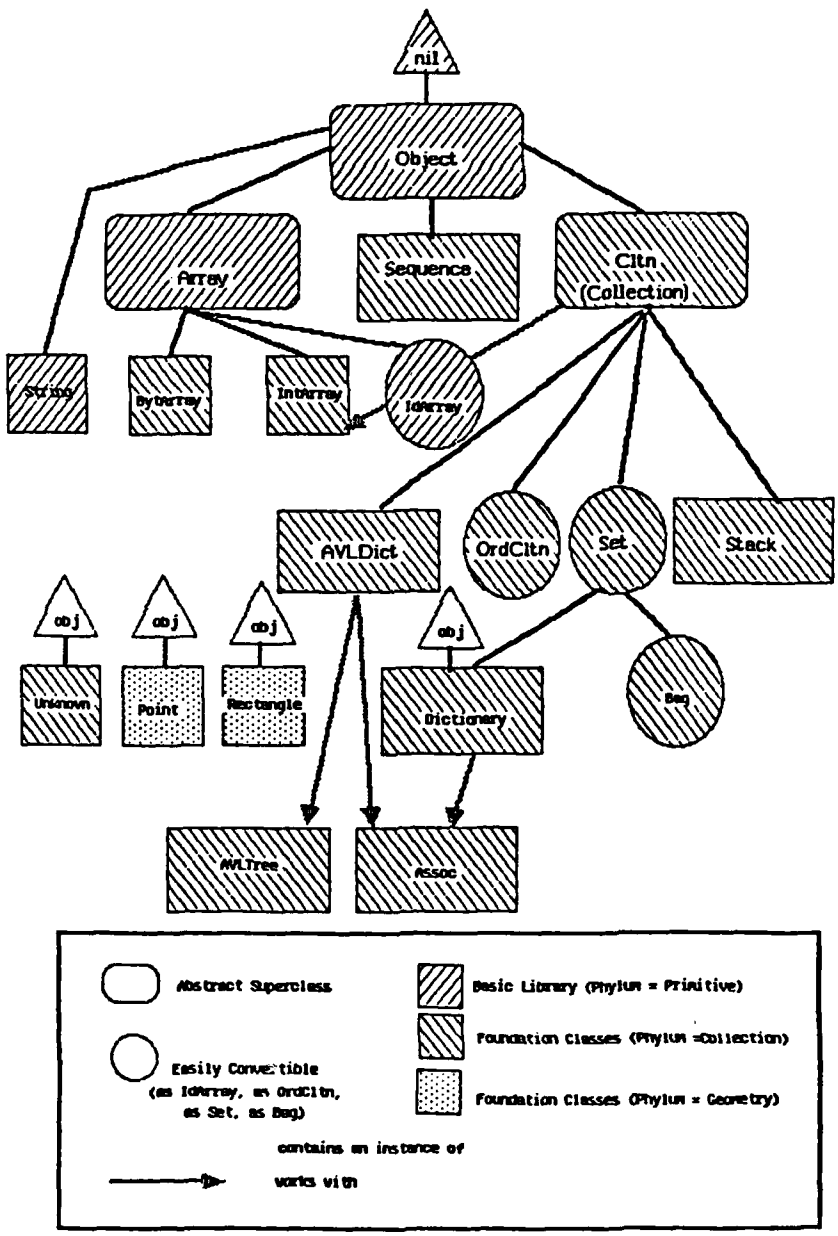
Objective-C Class Libraries

Included with the Objective-C compiler package are the Basic Class Library and the Foundation Class library that establish the root of the hierarchy of reusable classes from which classes for the specific application are developed. Classes developed for the application inherit properties of classes between the root and itself. The hierarchy of classes provided with the Objective-C compiler are presented graphically in appendix A.

The Basic Library contains the classes Nil, Object, Array, IdArray and String. The root of the inheritance hierarchy is class Object that points to the Nil class. Every object inherits all the methods and instance variable available in class Object. Class Array is detailed to give an idea of the methods this class supports. Array is a superclass of several classes that support indexed instance variables. It has an instance variable capacity that records the units of elements of the array. Methods are defined for instance creation with n-elements that may be initialized from an argument list or not. Methods are also defined for copying, inquiring on capacity, printing to an I/O device, comparing and hashing, and notifying on bounds violations.

The Foundation Library contains the classes Assoc, AVLDict, AVLTree, Bag, BytArray, Cltn, Dictionary, IntArray, Cltn, Dictionary, IntArray, OrdCltn, Point, Rectangle, Sequence, Sets, Stack and Unknown.

Diagram of Hierarchy of Classes in Basic and Foundation Library[PPI85]



The implementer of an Object/Message application must be familiar with the available classes to appropriately use the inheritance properties inherent in the class hierarchy. In the prototype implementation, the class Object was used. In the discussion of future work in Chapter 7, the development of other classes are described to support the task of creating consistency in the heterogeneous bibliographic citation database.

4.2.2 Unix Tool: Lex

The Unix tool, Lex, is a program or module generator. The basic model for Lex is based on the theory of regular expressions[Aho74]. It generates a module that is a deterministic finite state automaton. The input to Lex is based on user specified rules that are in the form of regular expressions. Regular expressions are rules for specifying character strings to be matched and include operator characters to account for repetition of strings, optional or required occurrences of strings, and the ordering of strings. The user may associate a procedure with a rule so further processing is done when a rule is matched. For example, if a rule in the form of a regular expression expects a number, the associated procedure may verify that the number is in an expected range and flag an error if it is not valid[Les75]. Lex generates the module that does lexical analysis on the input character stream consisting of the detail information associated with a data tag in a bibliographic citation. The tokens and optional values are passed to the parser.

4.2.3 Unix Tool: Yacc

Yacc is a tool that generates a program or module called the parser. Yacc is based on Context Free Grammars using Backus-Naur Form(BNF) descriptors to specify the parser that accepts the language. The formal discussion is found in [Aho74] and a user's manual in [Joh75]. The input to Yacc are user specified grammar rules and optional procedures to be invoked when the grammar rule is recognized. The parser includes a call to the lexical analyzer that passes tokens and optional values recognized from the input character stream.

The parser does a syntactic analysis and does the associated actions if the input satisfies the grammar rule. For the prototype the grammar rules include all the legal variations in the detail information for a data tag in a bibliographic citation.

4.2.4 Unix Tool: Make

The Unix tool Make is a software management tool that allows dependencies to be specified by the user among software modules. Changes to a source file are automatically detected and trigger the appropriate actions specified in the dependency rule. For example, modifications to a source file could trigger recompilations of other dependent source files.

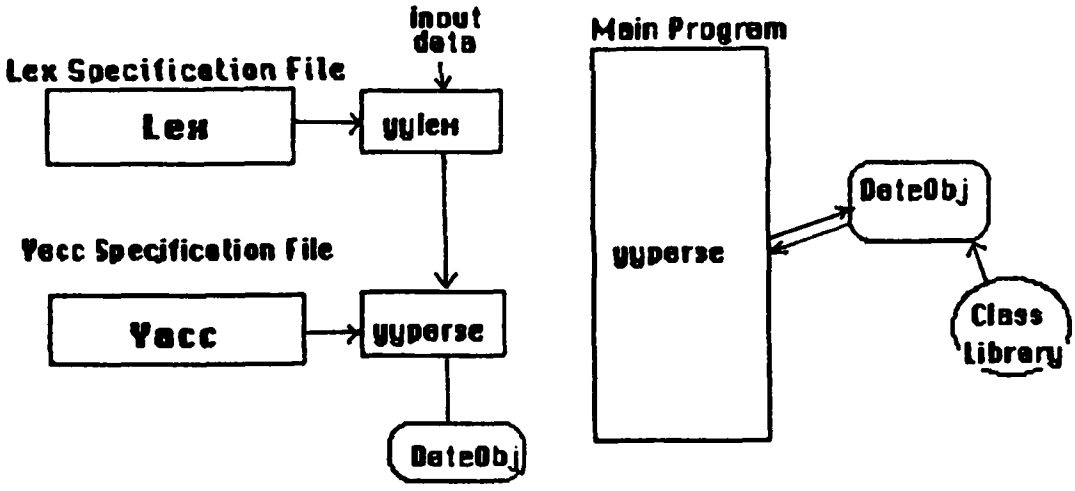
4.3 Summary

The software prototype was developed in the Unix BSD 4.2 software environment, using the Unix tools Lex, Yacc, Make and the Objective-C compiler. The C compiler was used to develop the software. The next chapter discusses implementation of the prototype and how the tools are used in the implementation.

Chapter 5:

Prototype Implementation

This chapter introduces the basic data abstraction mechanism in Objective-C, the class. A prototype for processing heterogeneous bibliographic information is described to show how the abstraction is used in program design and how it is used and implemented in Objective-C. A system overview that details the major steps in producing the prototype is diagramed.



System Overview of Prototype Implementation

5.1 Sources of Data

The source of data could be the result of a session by a user at a terminal making queries of an on-line system such as the Dialog system that involve the search of bibliographic citations on a topic. The output is usually in the form of a display of the retrieved citations and may be followed by a more complete printout of the citations. In our case, the facilities at the LLNL Technology Information System (TIS) were used to obtain bibliographic citations on the subject of "Computer Gateways and Networks" from the six following on-line database services: DTIC/DROLS-TR, DIALOG NTIS FILE 6, BRS, DOE/RECON, NASA/RECON, and SDC/LIBRARY and INFORMATION SCIENCE ABSTRACT. An on-line session with each particular database service was used to capture the information into a local file. The citations in the local file was translated into the TIS standard form for bibliographic citations. The six local files were then merged into a single file so that post-processing analysis could be done on a single file. A sample of the merged file is included in Appendix D.

Each bibliographic citation consists of an average of twenty fields of information. Each field begins on a new line and consists of a data tag delimited by left and right angle brackets (<,>) and ending with the descriptive information. In database terminology, one can consider the data tag as a field label and the descriptive information as the field detail.

5.2 Reformatting the Detail Information for Consistency

On closer examination of the bibliographic citations in the merged file one finds similar types of information may be represented in differing formats if they come from different database sources. There may be varying formats within a database for items coming from different publication types. For example, "<DATE> 1985." appears in a BRS/National Library of Medicine Database record, whereas, "<DATE> Aug 1984" appears in a DIALOG NTIS FILE 6 citation. Another problem is that "<TITLE> PLURIBUS SATELITE IMP DEVELOPMENT MOBILE ACCESS TERMINAL NETWORK" appears in upper-case in the DTIC/DROLS-TR citation but "<TITLE> An on-line directory of databases for material properties" appears in lower case except for the first word in the NASA/recon citation database. One can make the observation, however, that similar "classes" of information occur in bibliographic citations.

The task of reformatting the detail information for consistency is a complex job. The detail information from different database sources may appear with a different data tag. An example is "<TITLE> Post-processing of Bibliographic Citations from DOE/RECON, NASA/RECON, and DOD/DROLS. Revision 1." from the DIALOG NTIS FILE 6 whereas the same citation in the DOE/recon database has "<TITLE(MONO)> Post-processing of Bibliographic Citations from DOE/RECON, NASA/RECON, and DOD/DROLS. Revision 1." The task of consistency may include a cross correlation of information. If the title is not available with the <TITLE> data tag, the information may be available with the <TITLE(MONO)> data tag. Hence a duplicate may be detected and removed. Typically, one may request a yearly count of articles written on a subject to ascertain the emerging importance of research in the area. We pointed out in Chapter 2, they estimate that thirty-five percent of the bibliographic citations are duplicates[Hal83] and so the accounting of duplicates is important.

5.3 Program Design Abstractions

Consider the merged file as a data abstraction called in-stream, and the data abstraction called out-stream that will contain bibliographic citations in a consistent format. We will need procedural abstractions that indicate when in-stream is empty, or determine the next data tag and data field pair. We can consider each data tag and data field pair as an abstraction. Hence, we can arrive at abstract data types for "date", "title", "author", and etc. that are based on the data tags found in the merged file.

The <DATE> abstraction is presented with details for its implementation. The bibliographic data tags such as <DATE>, <AUTHOR>, or <TITLE> are handled as left context operators. They trigger environments that are very dissimilar. On closer examination, the information associated with <TITLE> is considered as a string, whereas the information associated with <DATE> is considered on a word basis, where a word is any nonempty sequence of alphanumeric characters. Adjacent words may be separated by non-alphanumeric characters as space, punctuation, or newline. Hence the lexical rules and actions must be specified separately for these two different environments. In looking at the <AUTHOR> and <DATE> detail information, the parser rules and actions must be specified individually also. An author June E. Smith has a first name of "June", whereas June should be handled as the sixth month if it is a date.

A discussion on handling of left context sensitivity is described in the Lex reference[Les75]. Once the data tag has been identified, then separate lexical and parser routines associated with Lex and Yacc rules are called to process the information. We can think of Lex and Yacc as procedural abstractions in the development of our prototype class. The Unix tools Yacc and Lex produce C modules of advanced algorithms in a convenient form that can be easily integrated into the prototype application program. These program generators do special jobs based on user specifications that are easy to update. Yacc produces the module "yyparse" and Lex produces the module "yylex". The user can insert C code before, within, and after the call to either module to add a large amount of flexibility. The modules generated are special purpose and have excellent performance in terms of time and space. They save the user from writing their own C code and hence frees the programmer from details that are conceptualized as procedural abstractions.

5.4 The Prototype

To show the ease in creating Objective-C classes, the prototype for the Date class is described. The prototype consists of the Lex and Yacc specification files, the Date class data abstraction, and the main program module. The tutorials on Lex and Yacc were helpful in developing the specification files[Bel78].

5.4.1 Lex Specification File

The general format of Lex input is:

```
{definitions}
%%
{rules}
%%
{user routines}
```

The definition section is:

```
%{
#include "objc.h"
#include "y.tab.h"
#define MON(x) { yylval.lex=x; return MONTH; }
=(N,Collection,Primitive)
%}
```

The include file "objc.h" contains most of the standard definitions for the user of the Objective-C compiler. The file contains various C types such as STR for string, SEL for selector, BOOL for boolean, IOD for I/O descriptor and SHR for the shared part of an object. The include file y.tab.h is created by Yacc and contains the tokens used for communication between the lexical analyzer and the parser. The macro MON(x) is defined to assign a value to yylval.lex that is returned to the parser. Values returned by the lexical analyzer and associated action procedures are integers by default. The rules to Yacc can define other types that the parser tree handles so the stack properly carries out the reduce and shifts to determine an accepting state for the statement being parsed. The Yacc discussion covers the union of types that account for the suffix ".lex". The last statement is an Objective-C declaration for the Phyla files.

The rules section consisting of regular expressions is:

```
%%
[jJ]an("."|uary)? MON(1);
---
[dD]ec("."|ember)? MON(12);
```

```
[0-9]          {yylval.lex = yytext[0] - '0' ; return DIGIT;}

[ ]            { ; /* delete blanks */ }

"\n"           { return EOL;      }

"."            { return EOL ; }
```

In the regular expression '[jJ]an("."|uary)?', the months are allowed in different forms, i. e. jan, jan., january,Jan, Jan., or January. The macro MON(x) is the action statement where the value returned is an integer, that is 1 for January, 2 for February, and etc. The value is stored in yylval.lex, and MONTH is the token returned. The characters 0 through 9 are recognized by the regular expression [0-9] and the action is to return the integer value for the character representation and DIGIT for the token. The regular expression [ ] deletes blanks since there is no action statement. The regular expression "\n" recognizes end-of-lines and returns the EOL token. The regular expression "." recognizes any other character and the action statement returns the single character.

The last section defines procedure "date(month,day,year)" for checking that the month is in the range 1-12, and the days for a month are correct. The leap year is taken into account on the days of a month. Terse error warnings are included that could be changed to more sophisticated error recovery actions. See Appendix C for the details. Hence the lexical analyzer module, yylex, should be able to recognize the tokens in the eight variations for "date" that are tabulated in Chapter 2.

5.4.2 Yacc Specification File

We now describe the specification file that is input to Yacc to generate the module yyparse. The general form looks like:

```
declarations

%%

rules
```

```
%%

programs
```

The declaration section is:

```
%{

#include "objc.h"

= (N, Collection, Primitive)

extern id dateObj;

%}

%union {

        short lex;

        id obj;

}

%Start  prog

%token<lex> DIGIT MONTH

%token<lex> EOL

%type<lex> number year day

%type<obj> DateStmt
```

In the declaration section we have the include file objc.h and the phyla declaration that were described in the previous section on Lex. The external declaration of the instantiation of the Date class, dateObj, is required since dateObj is created in the main program. The union statement defines the two data structures on the parser tree, the "lex" integer data structure and the Objective-C "obj" id data structure. The goal symbol, prog, is defined by the %Start statement, and the legal lexical tokens that yylex

recognizes are DIGIT, MONTH, and EOL. Number, year, and day are parsed by yyparse and have the "lex" integer data structure. The DateStmt has the "obj" id data structure.

The rules section is:

```
%%
prog: DateStmt EOL { exit ();} ;
DateStmt: MONTH day ',' year
            {
                date ( $1, $2, $4 );
                $$ = [dateObj mo: $1 da: $2 yr; $4 ];
                        [dateObj print];
            }
            | - - - }
day: number;
year: number;
number: DIGIT | number DIGIT {$$ = 10 * $1 + $2; };
```

The rules section specifies the BNF grammar for parsing the legal forms of date. The date procedure checks that the number of days is within the correct range for the month, with leap year taken into consideration.

The following statement:

```
$$ = [dateObj mo: $1 da: $2 yr: $4];
```

stores the month, day, and year values in the object, dateObj. The Objective-C message expression is contained between the pair of square brackets([...]). The message is sent to the receiver, dateObj. There

are three keyword selectors, mo, da, and year, that consist of a string of characters ending in a colon character. The arguments to the keyword selectors are $1, $2, and $4 that are obtained from the parse tree. This is an invocation of a method defined in the Date class and is a behavior in addition to the instance methods that Class Date inherits from the Object Class.

```
[dateObj print];
```

The print method is defined in the Date class and defines a behavior for printing the values stored in the dateObj object for month, day and year. The user simply invokes the print method and is not encumbered by the details of the data structures of month, day, or year to print the information correctly. In contrast, the Fortran programmer must know whether the month, day, or year may be in ascii, octal, or integer format to select the proper conversion specification in the "Format" statement. The proper definition of the methods in a class should encompass the create, modify, or reply so that the user's requirements in working with the class object is complete.

The program section is the last section and contains an error diagnostic that prints a warning to the user if the input can not be parsed by the grammar rules contained in the input specification file for Yacc. One may observe at this point how terse the software is to do all this work. The extraneous characters for space, /, and variations in the date format are handled with a minimum amount of software. The values for month, day, and year are stored as instance variables into the object, dateObj, through the method defined within the class Date, and the print operation is easily invoked since the details are encapsulated as a method in the class Date.

5.4.3 Date Class

The Date class is defined in the source code file, "date.m". The declaration section has the Objective-C include file, objc.h, and the Yacc include file, y.tab.h. Next, the declaration for ascii representations for month is included for the print method.

The following statement:

= Date:Object (N,Collection,Primitive)

reflects that the Date class inherits properties from the Object Class, and the Date class will be included in the writable phylum file "N". Also, the Date class may use the classes in the Objective-C libraries Collection, and Primitive. The instance variable are declared to be integer for month, day, and year, and are called mon, da, and yr respectively. The first method prefaced with "-mo: ..." stores the values in the instance object. The next method denoted by "-print ..." prints the date to the terminal. The print method will test for the default values of -1 and vary the printout. The three sample printout forms are:

1 May 1985

May 1985

1985

5.4.4 Main Module

The main program contained in the file, "main.m", begins with the include file for the C compiler standard I/O library, stdio.h, and the Objective-C include file, objc.h. The phyla declaration statement for the main program follows. The externals are declared in addition to the instance object, dateObj. The main program sets the output to be the terminal that is the Unix standard output device.

The statement:

dateObj = [Date new] ;

creates the object for the Date class. Since the method "new" is not defined in the Date Class, the method is inherited from the Object Class. The prompt ">" is printed at the terminal and then the input is expected from user at the terminal so that it can be parsed and have its values for month, day, and year stored into the date object just created. The print method is then invoked to verify the proper values are stored in dateObj for month, day, and year. The last two statements declare the classes and phyla that can be used in this application program.

Chapter 6

Summary and Results:

The intent of the prototype implementation is to provide a programming example of the Class data abstraction mechanism of Objective-C as applied to the Date class to obtain data consistency in varying forms of dates that are contained in bibliographic citations. Through a simple example, features of the abstraction mechanism in Objective-C have been presented. The Unix tools, Lex and Yacc were used to develop the procedural abstractions, yylex, and yyparse, that do the lexical analysis and syntactic analysis on the varying date forms. Eight variations of dates consisting of month, day and year were established in the dateObj object for the Date class. With the instance variables set to specific values, the print method could be invoked to take care of the task. The private data and data access methods are encapsulated within the Date class, and requires that the user communicate through messages to the object to elicit the behaviors desired.

The Date class is an elementary example to show how other classes for the bibliographic citation database can be developed for accomplishing data consistency in the numerous fields in a bibliographic citation. The Date class can easily be extended to included more methods, categorized as setting, inquiring, performing arithmetic and printing.

Setting:

1. -setmo: aMonth set the month
2. -setda: aDay set the day
3. -setyr: aYear set the year

Inquiring:

1. -getmo: aMonth reply the month
2. -getda: aDay reply the day
3. -getyr: aYear reply the year

Performing Arithmetic:

1. -julian reply the Julian day
2. -dayofyear reply the nth day of year

Printing

1. -printmo reply the month
2. -printdy reply the day
3. -printyr reply the year

The goal is to develop a comprehensive Date class to simplify the task of constructing reliable software that is easy to understand, modify, and maintain. This Date class will be part of the Class Library that is accessed by application programmers who will rely on the skill of the designer who develops the abstraction. The classes must be defined such that the behaviors of the class of information is fully defined. These include the create, modify and reply operations. In the event that additional behaviors are necessary, the concept of abstraction mechanisms in the programming language as Objective-C will guarantee that software will not have to be re-examined or re-written because of the change.

We briefly describe how the <AUTHOR> and <TITLE> classes can be defined and used in the application for data consistency in heterogeneous bibliographic citation databases. The main program is expanded to examine the in-stream of data and look for the "<AUTHOR>" or "<TITLE>" data tag. This is easily done since the data tags are enclosed in the left and right angle brackets. The characters following the right angle bracket are saved in a buffer until a left angle bracket is detected. This buffer of characters is then passed as data input to the parser developed for the particular data tag information.

In the TITLE data tag the Lex specification file will have the action statement convert the text to upper-case for consistency, and then will store the title into the object

```
yylval.obj = [String str: yytext];
return STRING;
```

The Yacc specification file will contain the action statement:

```
$$ = [titleObj str: $1 ] ;
```

In the case of the AUTHOR data tag, the buffer of characters captured after detecting the Author tag is passed to the Author parser that has BNF specifications to handle the variations in author names. The author list could be saved in the Set class. The creation of an Author object could include an initialization that would give a wild card character like "*" for the first or middle name in cases where the names are missing from the input stream. The methods defined for the author class could treat the names as wild cards when a match is required.

The next logical development is to define a citation object that contains the Author, Title, and Date Objects as a related triple.

```
extern id String, Set; id citationObj;
citationObj = [self with: 3
        [dateObj str];
        [titleObj str];
        [authorOjb str]; ];
```

Methods could be defined to create, add, delete, or modify a citation, in addition to printing the citation in "pretty" forms for easy user viewing.

The prime idea in defining classes for the heterogeneous bibliographic citation databases is to present the application programmer with abstractions that handle the data types involved, and include all methods to process the abstract data types. Hence the objects are the entities that are handled by the application programmer to reduce the details that must be remembered. The particular class should characterize the behavior of the data entirely. If not, additional methods may be added to the class definition. Indeed, even if this is done, software that has been written based on the former class definition may not have to be rewritten unless it accesses the new features in the class. The underlying physical structure of the program is taken care of by the physical interfaces used by the Objective-C compiler. The basic actions in programming the application are assignment statements that create objects and invocations of class methods through messages to the objects to exhibit behaviors.

Chapter 7

Discussion and Future Directions

In recent years a variety of powerful generic tools have been created. Database Management Systems(DBMS) and Spreadsheets are examples. They gain their power from the ability to operate on various data. They provide the generic operations of create, modify, and output. We have attempted to create the tool for data conversion. This study was restricted to bibliographic citations to see how far the idea of a generic library tool can be extended. The development of the generic library tool requires the definition of classes which the application programmer incorporates into user software. The concept of abstract data types via classes can be extended to Database Management Systems. If one considers the relational model, then the relations in the form of tables can be considered the data structure of the class. The operations of retrieve, update, and append with qualifiers can be considered the class methods. This abstraction is a convenient one for the application programmer since tables of information are a common occurrence. But a detail look at the physical implementation of the data structure may be complex. The storage and access mechanisms may be based on hashing algorithms if the data are sparse and have a balanced distribution. B-trees may be used with linked lists for fast searches. Here the user is relieved of the complexities that are left to the Database Management System implementers. To access the relations the user relies on the query language that allows operations on the relations. In this same regard, the person developing the classes for an Object Oriented application must provide the application programmer with the necessary classes to do a job. The classes must be general enough to handle application programs that have not yet been defined. This is what a good Database Management System provides, and is what the class library for the application should provide. Of course, Database Management Systems are always being enhanced to do a better job for the user, and it is expected that the class library will be improved with

time. What is important is that the user will not have to rewrite any software that has been developed. Even if the underlying physical structure is changed to improve speed or space, the user need not be concerned, and all the benefits will be automatically gained. One can now readily understand the strength in using abstractions. Through Object Oriented Programming the abstraction mechanism found in Database Management Systems and Spreadsheets can now be extended to programming languages through abstraction mechanisms provided in languages like Smalltalk-80 and Objective-C.

This project has demonstrated the feasibility of establishing data consistency in heterogeneous bibliographic citation databases through data abstractions, called classes. Future work involves specifying and implementing the full set of classes for this application. With the classes in place, the application programs can be written to further the data consistency goal.

We have discussed the bilio-citation object consisting of the title, author, and date objects. The objects associated within the citation object should be expanded to include the necessary elements for identifying a bibliographic citation. This requires the establishment of a canonical form for a bibliographic citation. A study of the bibliographic citation format from different sources shows that the data tag names are diverse and many are singular. For example, the DOE/RECON database uses "<PAGE NO> 17", whereas the DTIC/DROLS-TR has "<PAGINATION> 30P". Goldstein and Prettyman have proposed a set of 36 fields for the citation canonical form and it appears in chapter 2. They propose two character data tags, such as PG for the number of pages in the reference. Their canonical form is based on bibliography preparation. The data fields for the general case needs to be studied and proposed. On a cursory glance, the expanded canonical form should include "AB" for abstract, and "KW" for keyword descriptors. We note singular data tags that probably are only meaningful to the local bibliographic database such as "<LIMITATION CODES> 1", can be excluded from the canonical form of the citation. With the data tag and associated data elements defined for the canonical form of a bibliographic citation, the definition of classes for data consistency can proceed. The Date class can be re-used for the journal date, publication year, copyright year, and the meeting date. The definition of a Location class is appropriate for the meeting location, publication location, and author location. This class should access an abbreviation dictionary to produce a consistent form of the location.

If the location is listed as London, then London, England should be substituted. The location US, U.S.A., or United States should be made consistent in the same fashion. Warnings should be included for data not found in the dictionary, so that it may be updated with new entries. The standardization of publication titles can be added to a Source class. Certainly, the conversion for case consistency in a character strings, and the expansion of abbreviations should be included in the class methods. Alternate names for people or institutions could be accessible from a dictionary to further aid in data consistency. We note that the Dictionary class is available in Objective-C and can be incorporated into an class.

A future expansion should include the post-processing tasks in terms of the classes defined in the application tool library. Methods could be included to "pretty-print a bibliographic citation", to analyze bibliographic text, to display the citations on the CRT screen, to plot the statistical information on a graph, and to do cross-correlations on the data fields. The convenient tools of Unix can be incorporated into the classes since Objective-C is designed with the use of Unix tools in mind. We have seen how the Unix tools Lex and Yacc were incorporated into the Objective-C program.

The procedure of establishing data consistency in a heterogeneous bibliographic citation database through the definition of abstract data types can be extended to other heterogeneous databases. The restriction is that the information in the heterogeneous databases derive from a common base, as in bibliographic citations. Hence for a relational database where a relation is employee, a field in the relation is name, and its detail information is John Jones, the data tag could be <employee.name>, and the detail field would be John Jones. The existence of a data tag and and an associated detail field in the database establishes the reuse of the data abstractions created for the bibliographic citation database.

[Aho74] Aho, A. V., Hopcroft J. E., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.

[Bel78] *The Bell System Technical Journal*, July-August 1978, Vol. 57, No. 6, Part 2., American Telephone and Telegraph Co., pages 2155-2176.

[Bol84] Bollinger, W. A., Hampel, V. E., Harrison, I., Murphy,T.P., *Post-Processing of Bibliographic Citations from DOE/RECON, NASA/RECON, and DOD/DROLS*, Lawrence Livermore National Laboratory, UCRL-89995 Rev. 1, August 1984.

[Bur84] Burton, H. D., *Integration of an Automated Library Support System with an Intelligent Gateway*, Lawrence Livermore National Laboratory, UCRL-91383, August 1984.

[BuH84] Burton, H. D. and Hampel, V. E., *Integration of Common Command Languages with Intelligent Gateways*, Technology Information System, Lawrence Livermore National Laboratory, 1984.

[Cua84] Cuadra, R. N., Abels, D. M., Wagner, J, *Directory of Online Databases*, Cuadra Associates, Inc., Santa Monica, Ca., 1984, Vol. 5, No. 3, Spring 1984.

[Cox84] Cox, B. J., "Message/Object Programming: An Evolutionary Change in Programming Technology", *IEEE Software*, Vol. 1, Number 1, January 1984, pp50-61.

[Eag85] Eagles Project, Electronics Engineering, Engineering Research Division, Lawrence Livermore National Laboratory, Livermore, Ca., 1985.

[Hal83] Hall, J. L. and Brown, M. J., *Online Bibliographic Databases: A Directory and Sourcebook, Third Edition, 1983.*, Aslib, London, 1983.

[Ham79] Hampel, V. E., McGrogan, S. K., Gallo, L. E., Swanson, J. E., *The LLNL "Meta-Machine"*, Fourth Berkeley Conference on Distributed Data Management and Computer Networks, San Francisco, California, August 28-30, 1979, Lawrence Livermore National Laboratory, UCRL-

83064, May, 1979.

[Ham85] Hampel, V. E., "TIS, The Intelligent Gateway Processor", *Proceedings of the Eighteenth Annual Hawaii International Conference on System Sciences*, 1985.

[Gol83] Goldberg, A. and Robson, D., *Smalltalk-80, The Language and its Implementation*, Addison-Wesley, New York, 1983.

[Gol84] Goldberg, A. *Smalltalk-80, The Interactive Programming Environment*, Addison-Wesley, New York, 1984.

[Gol85] Goldstein, C. M. and Prettyman, M., Processing Downloaded Citations", Lister Hill National Center for Biomedical Communications, National Library of Medicine, Bethesda, Md., 1985.

[KeP84] Kernighan, B. W., Pike, R., *The Unix Programming Environment*, Prentice-Hall Software Series, Englewood Cliffs, N.J., 1984.

[Joh75] Johnson, S. C., *Yacc: Yet Another Compiler Compiler*, Computing Science Technical Report No. 32, 1975, Bell Laboratories, Murray Hill, New Jersey, 1975.

[Les75] Lesk, M. E. and Schmidt, E., *Lex- A Lexical Analyzer Generator*, Computing Science Technical Report No. 32, 1975, Bell Laboratories, Murray Hill, New Jersey, 1975.

[Lis74] Liskov, Barbara, Zilles, Stephen, *Programming with Abstract Data Types*, Proc. ACM SIGPLAN Conf. on Very High Level Language., SIGPLAN Notice 9,4 (April 1974) 50-59.

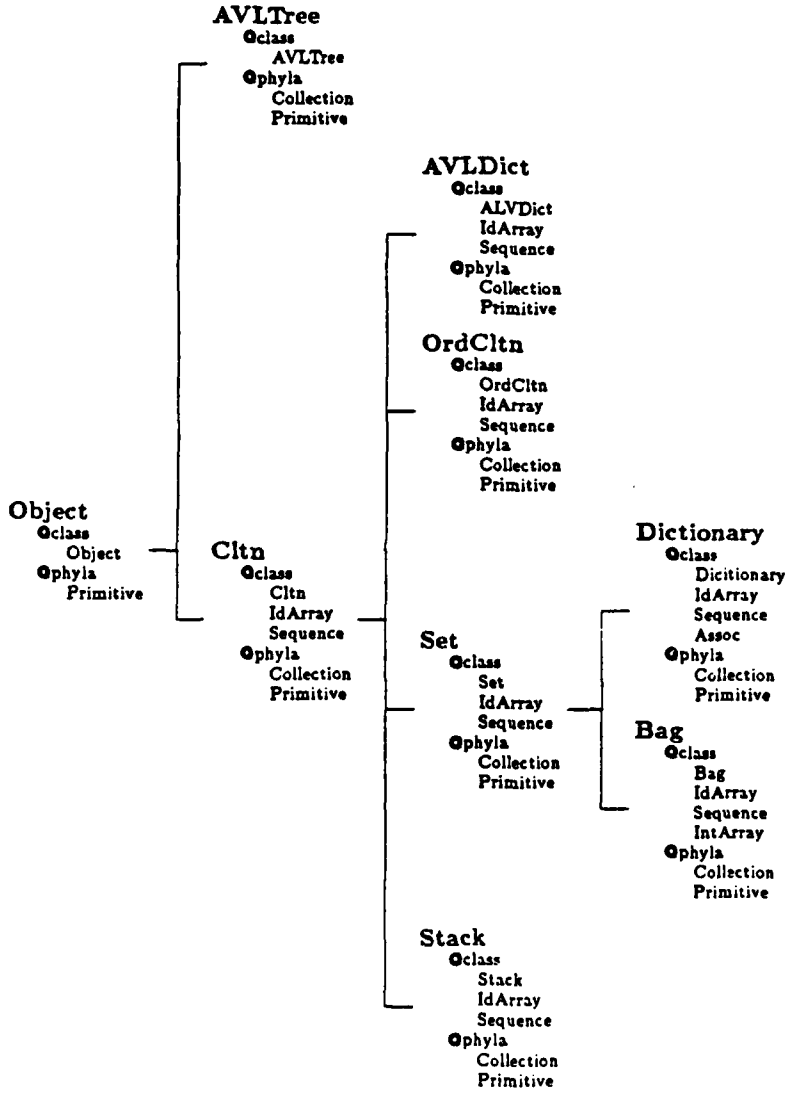
[Lis77] Liskov, Barbara, Snyder, A., Atkinson, R., and Schaffert, C., *Abstraction Mechanisms in CLU*, Comm ACM, 20, 8, August 1977, 564-576.

[PPI85] *Objective-C Reference Manual*, Productivity Products International, Sandy Hook, CT, 1985.

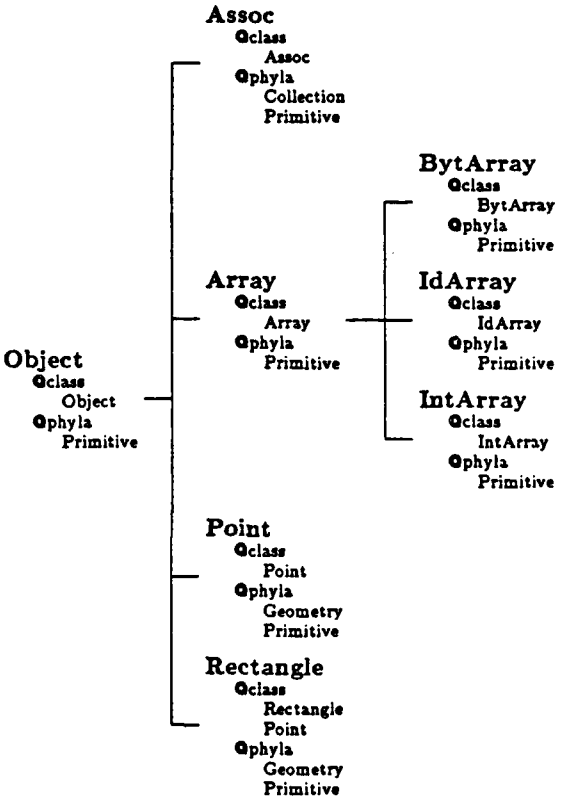
[Sha84] Shaw, Mary, *Abstraction Techniques in Modern Programming Languages*, IEEE Software, Oct. 1984.

Appendix A

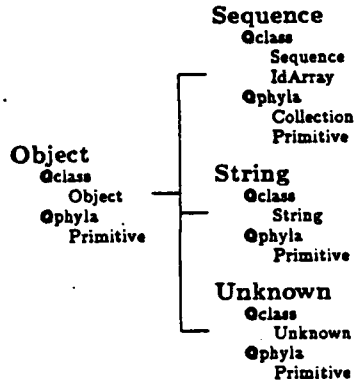
Hierarchy of Objective-C Classes - @class, @phyla [PPI85][Eag85]



Hierarchy of Objective-C Classes (continued)

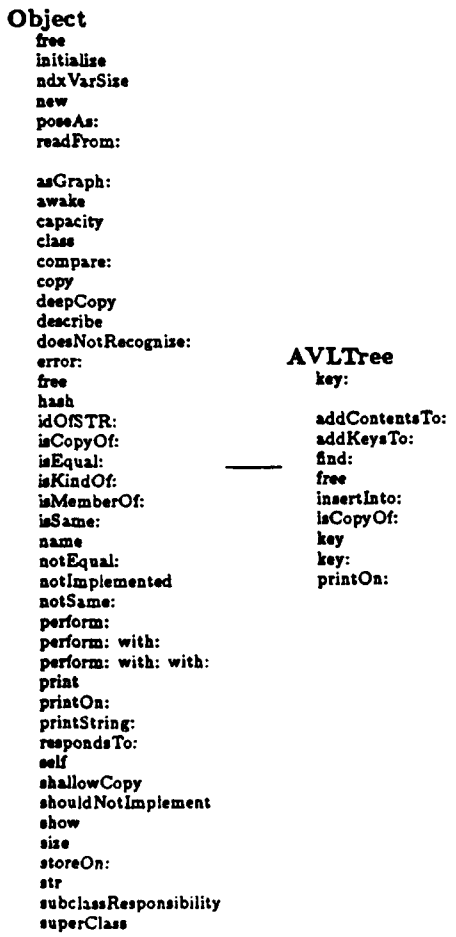


Hierarchy of Objective-C Classes (continued)

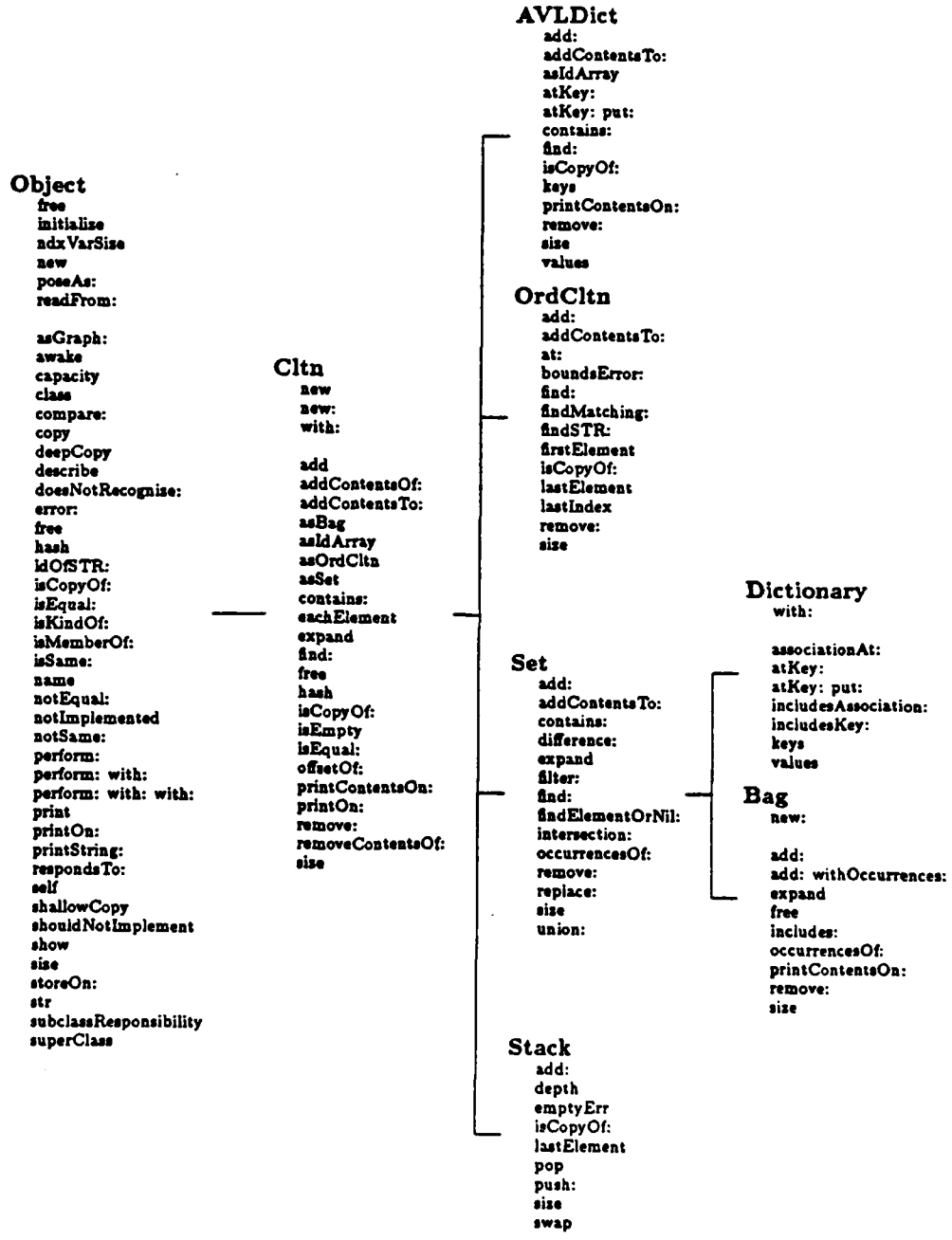


Appendix B

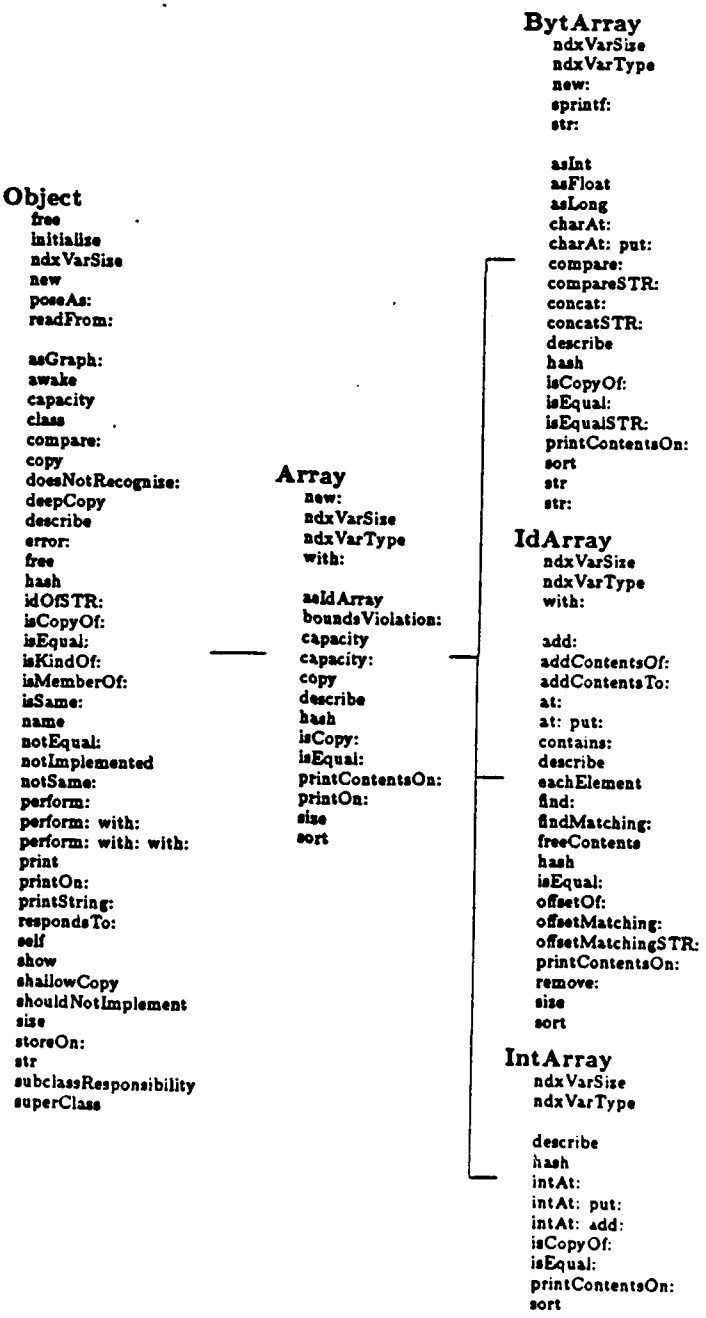
Objective-C Base Tree - methods [PPI85][Eag85]



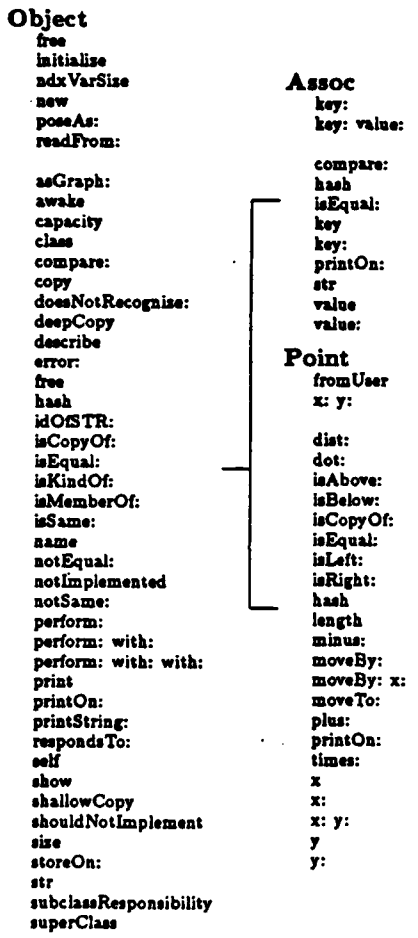
Objective-C Base Tree (continued)

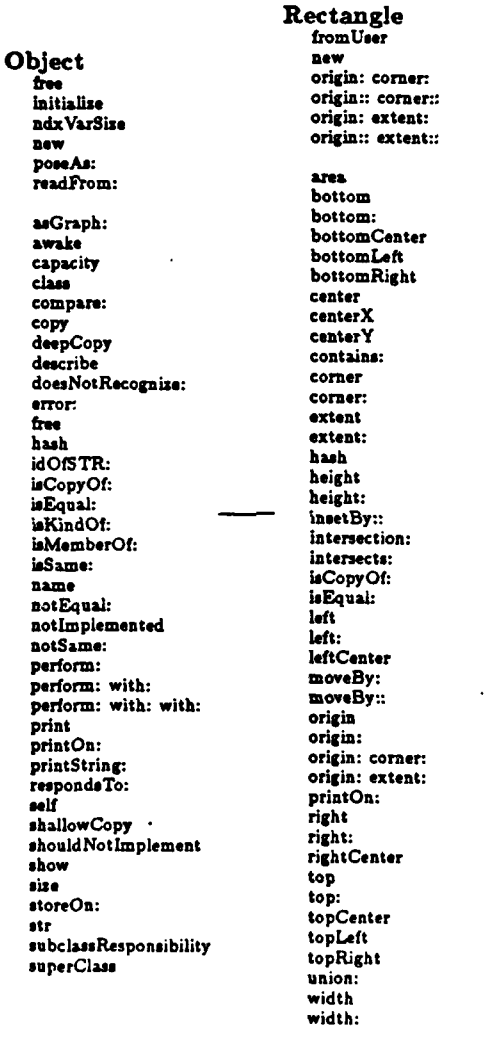


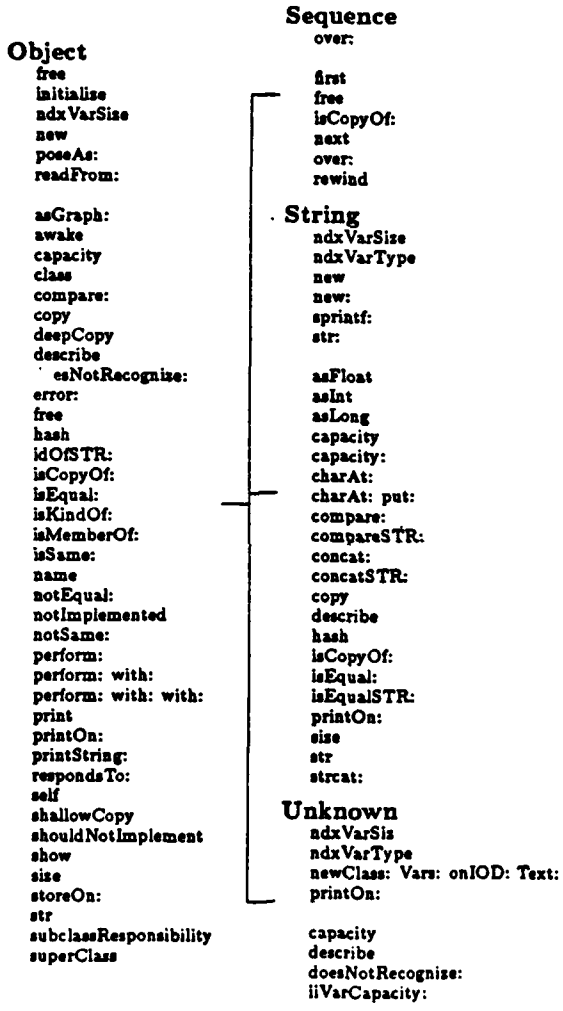
Objective-C Base Tree (continued)



Objective-C Base Tree (continued)




Object
free
initialize
ndxVarSize
new
poseAs:
readFrom:
asGraph:
awake
capacity
class
compare:
copy
deepCopy
describe
doesNotRecognize:
error:
free
hash
idOfSTR:
isCopyOf:
isEqual:
isKindOf:
isMemberOf:
isSame:
name
notEqual:
notImplemented
notSame:
perform:
perform: with:
perform: with: with:
print
printOn:
printString:
respondsTo:
self
shallowCopy
shouldNotImplement
show
size
storeOn:
str
subclassResponsibility
superClass
Rectangle
fromUser
new
origin: corner:
origin:: corner::
origin: extent:
origin:: extent::
area
bottom
bottom:
bottomCenter
bottomLeft
bottomRight
center
centerX
centerY
contains:
corner
corner:
extent
extent:
hash
height
height:
insetBy::
intersection:
intersects:
isCopyOf:
isEqual:
left
left:
leftCenter
moveBy:
moveBy::
origin
origin:
origin: corner:
origin: extent:
printOn:
right
right:
rightCenter
top
top:
topCenter
topLeft
topRight
union:
width
width:


Object
free
initialize
ndxVarSize
new
poseAs:
readFrom:
asGraph:
awake
capacity
class
compare:
copy
deepCopy
describe
esNotRecognize:
error:
free
hash
idOfSTR:
isCopyOf:
isEqual:
isKindOf:
isMemberOf:
isSame:
name
notEqual:
notImplemented
notSame:
perform:
perform: with:
perform: with: with:
print
printOn:
printString:
respondsTo:
self
shallowCopy
shouldNotImplement
show
size
storeOn:
str
subclassResponsibility
superClass
Sequence
over:
first
free
isCopyOf:
next
over:
rewind
String
ndxVarSize
ndxVarType
new
new:
sprintf:
str:
asFloat
asInt
asLong
capacity
capacity:
charAt:
charAt: put:
compare:
compareSTR:
concat:
concatSTR:
copy
describe
hash
isCopyOf:
isEqual:
isEqualSTR:
printOn:
size
str
strcat:
Unknown
ndxVarSiz
ndxVarType
newClass: Vars: onIOD: Text:
printOn:
capacity
describe
doesNotRecognize:
iiVarCapacity:

Appendix C

Prototype Source Code

LEX Specification File

```
%{
#include "objc.h"
#include "y.tab.h"
#define MON(x) { yylval.lex = x ; return MONTH ; }
= (N, Collection, Primitive)
%}
%%
[jJ]an("."|uary)?  MON(1) ;
[fF]eb("."|ruary)? MON(2) ;
[mM]ar("."|ch)?    MON(3) ;
[aA]pr("."|il)?    MON(4) ;
[mM]ay             MON(5) ;
[jJ]un("."|e)?     MON(6) ;
[jJ]ul("."|y)?     MON(7) ;
[aA]ug("."|ust)?   MON(8) ;
[sS]ep("."|tember)? MON(9) ;
[oO]ct("."|ober)?   MON(10) ;
[nN]ov("."|ember)?  MON(11) ;
[dD]ec("."|ember)?  MON(12) ;
[0-9]           { yylval.lex = yytext[0] - '0' ;
                  return DIGIT ;                 }
[ ]             { ; /* delete blanks */ }
"\n"            { return EOL ; }
"."             { return EOL ; }
.                {
                   return  (yytext[0]) ;   }  /* return single characters */
%%
# include "stdio.h"
int noleap [] = {
                    0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, } ;
int   leap [] = {
                    0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, } ;
date ( month, day, year )
 {
  int *daysin ;
  daysin = isleap ( year ) ? leap : noleap ;
  if (month < 1 ||  month > 12 )
     { printf ( "month out of range \n" )  ;
       return ;
     }
  if ( day < 1 || day > daysin[month] )
     { printf ( "day of the month out of range\n" ) ;
       return ;
     }
 }
isleap (year)
 { if (year % 4   != 0 ) return (0) ;
   if (year % 100 != 0 ) return (1) ;
   if (year % 400 != 0 ) return (0) ;
                         return (1) ;
}
int yywrap ()
{return(1); }
```

YACC Specification File

```
%{
#include "objc.h"
- (N, Collection, Primitive)      /*  phyla  */
extern id dateObj ;
%}
%union  {                         /* stack type */
        short lex ;               /* lexical code */
        id obj ;                  /* an object */
}
%Start      prog
%token<lex> DIGIT MONTH
%token<lex> EOL
%type<lex>  number year day
%type<obj>  DateStmt
%%
prog:   DateStmt  EOL { exit(); } ;
DateStmt: MONTH day ',' year
          {
           date ( $1, $2, $4 ) ;
           $$ = [dateObj mo: $1 da: $2 yr: $4 ] ;
                [ dateObj print ] ;
          }
        | day MONTH year
          {
           date ( $2, $1, $3) ;
           $$ = [ dateObj mo: $2 da: $1 yr: $3 ] ;
                [ dateObj print ] ;
          }
        | number '/' number '/' number
          {
           if ($5 < 100) $5 = $5 + 1900;
           date ( $1, $3, $5 ) ;
           $$ = [ dateObj mo: $1 da: $3 yr: $5 ] ;
                [ dateObj print ] ;
          }
        | number '.'
          {
           date ( 1, 1, $1) ;
           $$ = [ dateObj mo: -1 da: -1 yr: $1 ] ;
                [ dateObj print ] ;
          }
        | number
          {
           date (1, 1, $1) ;
           $$ = [ dateObj mo: -1 da: -1 yr: $1 ] ;
                [ dateObj print ] ;
          }
        | MONTH number
          {
           date ( $1, -1, $2) ;
           $$ = [ dateObj mo: $1 da: -1 yr: $2) ;
                [ dateObj print ] ;
          }
      |  MONTH ',' number
          {
           date ( $1, -1, $3 ) ;
           $$ = [ dateObj mo: $1 da: -1 yr: $3) ;
                [ dateObj print ] ;
          }
        ;

day:      number
        ;
year:     number
        ;
number:   DIGIT
        | number DIGIT
          { $$ = 10 * $1 + $2 ; }
        ;


%%
#include "stdio.h"
yyerror (s)   /* called for yacc syntax error */
        char *s;
{
        warning(s, (char *)  0);
}
        char *progname="stdin";
warning(s, t1, t2, t3, t4, t5, t6, t7, t8, t9)  /*  print warning message */
        char *s, *t1, *t2, *t3, *t4, *t5, *t6, *t7, *t8, *t9;
{
        extern int yylineno;
/*      fprintf(stderr, "file %s: ", progname); */
        fprintf(stderr, s, t1, t2, t3, t4, t5, t6, t7, t8, t9);
        fprintf(stderr, " near line %d\n", yylineno);
}
```

```
#include "objc.h"
#include "y.tab.h"
char * MON[] = { " ","Jan", "Feb", "Mar", "Apr", "May", "Jun",
          "Jul", "Aug", "Sep", "Oct", "Nov", "Dec", } ;
= Date:Object ( N, Collection, Primitive )
 { int mon, dy, year ; }
- mo:(int) aMonth da:(int) aDay yr:(int) aYear
 {   mon = aMonth ;
     dy = aDay ;
     year = aYear ;
     return self ;
 }
- print
  {
   if (dy>0 && mon>0 )
   printf("<DATE> %d %s %d\n", dy, MON[mon], year ) ;

   if (dy < 0 && mon<0 )
   printf("<DATE> %d\n", year ) ;

   if (dy < 0 && mon >0 )
   printf ("<DATE> %s %d\n", MON[mon], year ) ;

/*    insert code for different type of prints to account for defaults*/
  }
```

Main Program Source File

```
#include "stdio.h"
#include "objc.h"
= ( N, Collection, Primitive )
  extern BOOL msgFlag ;
  extern IOD  yyin, yyout, msgIOD ;
  id dateObj ;

main ()
 {

  extern id  Date, Set ;



  msgIOD = stdout;
  msgFlag = NO ;

  dateObj = [ Date new ] ;

  printf (">" ) ;
  yyparse() ;

  printf ("end yyparse\n");
 }

@class ( Date, Set,Cltn, IdArray, Sequence )
@phyla (N, Collection, Primitive )
```

Appendix D

Merged File of Heterogeneous Bibliographic Citations from Six Database Sources

<ACCESSION NO.> 85129022 8506.
<DATABASE SOURCE> BRS/National Library of Medicine Database
<AUTHORS> Ellison-J-M; Wharff-E-A;
<PAA> Cambridge Hospital, Massachusetts.
<TITLE> More than a gateway: the role of the emergency psychiatry service in the community mental health network.
<PUB DESC> Hosp-Community-Psychiatry. 1985 Feb. 36(2). P 180-5.
<LANGUAGE> EN.
<MAJOR CATEGORY> COMMUNITY-MENTAL-HEALTH-CENTERS: og. EMERGENCY-SERVICE-HOSPITAL. EMERGENCY-SERVICES-PSYCHIATRIC og. INTERINSTITUTIONAL-RELATIONS. MENTAL-HEALTH-SERVICES og.
<MINOR CATEGORY> ADULT. BOSTON. CASE-REPORT. CATCHMENT-AREA-HEALTH CRISIS-INTERVENTION. FEMALE. HOSPITAL-BED-CAPACITY-300-TO-499. HUMAN. MALE MIDDLE-AGE. ROLE SOCIAL-WORK or in helping the emergency unit build closer relationships with community agencies is its contract with the state to perform evaluations of all admissions to the state hospital psychiatric unit serving the catchment area. The emergency unit performs triage and provides backup for the agencies, coordinates the management of multi-agency cases, and holds weekly educational conferences for agency staff. Using case examples, the authors illustrate how unit and agency staff collaborate to ensure continuity of patient care. Author.
<SB> M
<DATE> 1985.
<ISSN> 0022-1597.
<ZN> Z1 107.567.875..
<!M> 8506.
<ED> 850404
<NO> MH17582.
<ACCESSION NO.> A147675 - 1
<DATABASE SOURCE> DTIC/drols-tr
<TRANSLATION DATE> Mon Jul 1 13:33:43 PDT 1985 (489098023)
<DOWNLOAD DATE> Mon Jul 1 10:18:29 PDT 1985 (489086309)
<DOWNLOAD FILE NAME> gate
<FIELDS AND GROUPS> 17/2
<ENTRY CLASSIFICATION> UNCLASSIFIED
<CORPORATE AUTHOR> BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA
<TITLE> PLURIBUS SATELITE IMP DEVELOPMENT MOBILE ACCESS TERMINAL NETWORK.
<TITLE CLASSIFICATION> UNCLASSIFIED
<DESCRIPTIVE NOTE> QUARTERLY TECHNICAL REPT. NO. 33, 1 FEB-30 APR 84.
<DATE> MAY , 1984
<PAGINATION> 30P
<REPORT NUMBER> BBN-5774
<CONTRACT NUMBER> MDA903-80-C-0353, N00039-81-C-0408
<REPORT CLASSIFICATION> UNCLASSIFIED
<DESCRIPTORS> *SATELLITE COMMUNICATIONS; *TERMINALS; NETWORKS; SHIPBOARD; ACCESS; MOBILE; WORK
<DESCRIPTOR CLASSIFICATION> UNCLASSIFIED
<IDENTIFIERS> PLURIBUS SATELLITE, PACKET NETWORKS, ARPANET, GATEWAYS
<IDENTIFIER CLASSIFICATION> UNCLASSIFIED
<ABSTRACT> THIS QUARTERLY TECHNICAL REPORT DESCRIBES WORK ON THE DEVELOPMENT OF PLURIBUS SATELLITE IMPS: AND ON SHIPBOARD SATELLITE COMMUNICATIONS. (AUTHOR)
<ABSTRACT CLASSIFICATION> UNCLASSIFIED
<INITIAL INVENTORY> 12
<LIMITATION CODES> 1
<SOURCE CODE> 060100
<DOCUMENT LOCATION> NTIS
<GEOPOLITICAL CODE> 2508
<TYPE CODE> 4
<ACCESSION NO.> 1103508
<DATABASE SOURCE> DIALOG NTIS FILE 6
<REPORT NO.> <NTIS> DE85000617/XAB
<TITLE> Post-Processing of Bibliographic Citations from DOE/RECON, NASA/RECON, and DOD/DROLS. Revision 1
<AUTHORS> Ballinger, W. A. , Hampel, V. E. ; Harrison, I. , Murphy, T. P.
<PUB DESC> Lawrence Livermore National Lab., CA. ; <Code> 868147000; 9513035 ; Department DC. ; UCRL-89995-REV.1; CONF-841243-1-REV.1
<DATE> Aug 1984
<PG> 17p
<LANGUAGE> English
<DOCUMENT TYPE> Conference proceeding
<PC> PC A02/MF A01
<JA> GRA18507; NSA1000
<CO OF PUBL> United States
<NT> International online information meeting, London, UK, 4 Dec 1984.
<CN> W-7405-ENG-48
<ABSTRACT> We have developed an interactive, self-guided program for the

joint post-processing of bibliographic citations from the federal information centers of the Department of Energy (DOE), the Department of Defense (DOD), and the National Aeronautics and Space Administration (NASA). This program is currently installed on the Intelligent Gateway Processor of the Technology Information System (TIS/IGP) at the Lawrence Livermore National Laboratory and is under evaluation by the TIS user community from remote terminals by telephone dial-up, over TYMNET, and the ARPA computer network. Users are individually authorized for automated access to specific information centers, and use standard commands for the downloading, compilation, and online review of citations in a common format. Previously reported post-processing capabilities have been further expanded, permitting: (1) online citation review, categorization, and addition of new data elements; (2) disassembly and re-assembly of citations; (3) statistical analysis of data field contents; (4) cross-correlation of data field contents; and (5) concordance generation. In addition, the new two-pass interpreter for the post-processing program permits: the transformation of abbreviated data field names into English names preferred by each agency, the statistical analysis of the density and completeness of data fields in selected sets of bibliographic citations, the elimination of redundant citations (using user-specified criteria), and trend analysis. The latter is a powerful tool for the exploration of time-dependent characteristics in a particular field of research, of an organization, or for an author. Graphical displays of publication rates as a function of time and the normalized statistics of terms used in the description of the work, can be used to signal new directions of ongoing research and the intensity of its support. (ERA citation 10:001706)
<DESCRIPTORS> *Information; *Computer Networks; Information Retrieval; Specifications
<Indexing Terms> ERDA/990300; NTISDE
<SH> 58 (Behavioral and Social Sciences--Documentation and Information Technology); 98 (Electronics and Electrical Engineering--Computers); 88B (Library and Information Sciences--Information Systems); 62B (Computers, Control, and Information Theory--Computer Software)
<ACCESSION NO.> 84C0188550
<DATABASE SOURCE> DOE/recon
<TRANSLATION DATE> Mon Jul 1 13:33:43 PDT 1985 (489098023)
<DOWNLOAD DATE> Mon Jul 1 10:18:29 PDT 1985 (489086309)
<DOWNLOAD FILE NAME> gate
<REPORT NO,PAGE> UCRL--89995-Rev.1 P. 17;DE85000617
<TITLE(MONO)> Post-processing of bibliographic citations from DOE/RECON, NASA/RECON, and DOD/DROLS. Revision 1
<EDITOR OR COMP> Bollinger, W.A.; Hampel, V.E.; Harrison, I.; Murphy, T.P.
<CORPORATE AUTH> Lawrence Livermore National Lab., CA (USA)
<CORPORATE CODE> 9513035
<TYPE> R
<SEC REPT NO> CONF-841243--1-Rev.1
<PAGE NO> 17
<AVAILABILITY> NTIS, PC A02/MF A01.
<ORDER NUMBER> DE85000617
<CONTRACT NO> Contract W-7405-ENG-48
<CONF TITLE> 8. international online information meeting
<CONF PLACE> London, UK
<CONF DATE> 4 Dec 1984
<DATE> Aug 1984
<CO OF AUTH> US
<CO OF PUBL> US
<ANN J> ERA-10:001706;EDB-84:188555
<DISTRIBUTION> MN-32
<DOCUMENT ORIGIN> P
<BIS> TIC
<CATEGORIES> EDB-990300
<PRIMARY CAT> EDB-990300(GENERAL AND MISCELLANEOUS; INFORMATION HANDLING)
<ABSTRACT> We have developed an interactive, self-guided program for the joint post-processing of bibliographic citations from the federal information centers of the Department of Energy (DOE), the Department of Defense (DOD), and the National Aeronautics and Space Administration (NASA). This program is currently installed on the Intelligent Gateway Processor of the Technology Information System (TIS/IGP) at the Lawrence Livermore National Laboratory and is under evaluation by the TIS user community from remote terminals by telephone dial-up, over TYMNET, and the ARPA computer network. Users are individually authorized for automated access to specific information centers, and use standard commands for the downloading, compilation, and online review of citations in a common format. Previously reported post-processing capabilities have been further expanded, permitting: (1) online citation review, categorization, and addition of new data elements; (2) disassembly and re-assembly of citations; (3) statistical analysis of data field contents; (4) cross-correlation of data field contents; and (5) concordance generation.

```
  In addition, the new two-pass interpreter for the post-processing program
  permits: the transformation of abbreviated data field names into english
  names preferred by each agency, the statistical analysis of the density
  and completeness of data fields in selected sets of bibliographic
  citations, the elimination of redundant citations (using user-specified
  criteria), and trend analysis. The latter is a powerful tool for the
  exploration of time-dependent characteristics in a particular field of
  research, of an organization, or for an author. Graphical displays of
  publication rates as a function of time and the normalized statistics of
  terms used in the description of the work, can be used to signal new
  directions of ongoing research and the intensity of its support.
<DESCRIPTORS> *INFORMATION--computer networks;INFORMATION RETRIEVAL.
  SPECIFICATIONS
<ISSUE> 8423
<DOCUMENT NO> 84:188555
<ACCESSION NO> B4C017369?
<DATABASE SOURCE> DOE/recon
<TRANSLATION DATE> Mon Jul 1 13:33:43 PDT 1985 (489098023)
<DOWNLOAD DATE> Mon Jul 1 10:18:29 PDT 1985 (489086309)
<DOWNLOAD FILE NAME> gate
<REPORT NO,PAGE> UCRL--91383 P. 10;DE85001741
<TITLE(MONO)> Integration of an automated library support system with an
  intelligent gateway
<EDITOR OR COMP> Burton, H.D.
<CORPORATE AUTH> Lawrence Livermore National Lab., CA (USA)
<CORPORATE CODE> 9513035
<TYPE> R
<SEC REPT NO> CONF-8409138--1
<PAGE NO> 10
<AVAILABILITY> NTIS, PC A02/MF A01.
<ORDER NUMBER> DE85001741
<CONTRACT NO> Contract W-7405-ENG-48
<CONF TITLE> Integrated online library systems conference
<CONF PLACE> Atlanta, GA, USA
<CONF DATE> 13 Sep 1984
<DATE> Aug 1984
<CO OF AUTH> US
<CO OF PUBL> US
<ANN J> EDB-84-173691
<DISTRIBUTION> MN-32
<DOCUMENT ORIGIN> P
<BIS> TIC
<CATEGORIES> EDB-990300
<PRIMARY CAT> EDB-990300(GENERAL AND MISCELLANEOUS: INFORMATION HANDLING)
<ABSTRACT> A new project of the Technology Information System (TIS) at the
  Lawrence Livermore National Laboratory (LLNL) calls for the evaluation of
  commercially available library support packages and the extension and
  integration of the most desirable system with the TIS gateway to provide a
  comprehensive prototype for libraries and information centers. This
  prototype system is planned to facilitate access to and management of
  in-house activities such as cataloging, serials control, and acquisitions,
  as well as to interface to external systems and services for data
  downloading and exchange, retrieval, and post-processing. Cooperative
  cataloging, distributed database processing, electronic inter-library
  loan, and customized bibliography production are some of the features
  planned for the prototype. Development of a user-friendly front-end
  processor will allow the user to negotiate his search query in a
  semi-automated manner using a single, English-like command language. The
  TIS at Lawrence Livermore National Laboratory (LLNL) has developed a
  computer-based intelligent gateway for automated access to such diverse,
  geographically distributed information systems as DOE/RECON, DOD/DROLS,
  NASA/RECON, CAS On-Line, DARC (France) and DECHEMA (West Germany), among
  many others. New information resources centers are being added as required
  and users can connect simultaneously to more than one host to compare
  their data. The TIS online master directory provides the user with a
  single, integrated view of available and relevant resources. The automated
  access procedures permit the user to concentrate on the information
  aspects of his work rather than be burdened with various log-on
  procedures, database formats and protocols. The merger of the library
  support with the TIS gateway should provide users with a capabilities to
  access and utilize the full spectrum of textual, numeric and graphics data
  resources
<DESCRIPTORS> *INFORMATION SYSTEMS--computer networks;DATA BASE MANAGEMENT;
  LAWRENCE LIVERMORE LABORATORY
<ISSUE> 8421
<UPPOSTED DESC> MANAGEMENT;NATIONAL ORGANIZATIONS;US AEC;US DOE;US ERDA;US
  ORGANIZATIONS
<DOCUMENT NO> 84-173691
```

<ACCESSION NO > 84N33099#
<DATABASE SOURCE> NASA/recon
<TRANSLATION DATE> Mon Jul 1 13:33:43 PDT 1985 (489098023)
<DOWNLOAD DATE> Mon Jul 1 10:18:29 PDT 1985 (489086309)
<DOWNLOAD FILE NAME> gate
<ISSUE> 22
<PAGE> 3643
<CATEGORY> 62
<RPT#> DE84-013210 UCRL-90276 CONF-8406139-1
<CNT#> W-7405-ENG-48
<DATE> 1984
<PAGES> 122
<DOC. CLASSIF.> UNCLASSIFIED
<TITLE> An online directory of databases for material properties
<AUTHORS> HAMPEL, V. E.; BOLLINGER, W. A.; GAYNOR, C. A.; OLDANI, J. J.
<PAA> C/(Control Data Corp.)
<PUB DESC> California Univ., Livermore. Lawrence Livermore Lab. CSS:
(Technology Information System.) AVAIL.NTIS SAP: HC A06/MF A01 Presented
at the 9th Intern. CODATA Conf., Jerusalem, 24-28 Jun. 1984
<DESCRIPTORS> DATA BASE MANAGEMENT SYSTEMS;DATA BASES;DIRECTORIES;INFORMATION
DISSEMINATION;INFORMATION SYSTEMS
<MINS> / COMPUTER NETWORKS/ COMPUTER TECHNIQUES/ DATA PROCESSING/ ON-LINE
SYSTEMS / STATISTICAL ANALYSIS
<ABA> DOE
<ABSTRACT> An online directory of databases of material properties on the
Technology Information System at Lawrence Livermore National Laboratory
(LLNL/TIS) is described. This directory is intended to provide interactive
access to scientific and technical databases available to the public that
contain information pertaining to nuclear, atomic, molecular, physical,
chemical, and mechanical properties of substances. In addition to the 101
data files previously are reported. The information is updated with more
than 38 numeric databases and predictive systems in these fields. In
addition to describing the contents of the databases, updated information
is provided on the availability of the databases and their online access
over public telephone and data networks. Some of the numeric databases are
directly accessible by authorized users via the TIS Intelligent •• Gateway
•• Processor at LLNL (TIS/IGP), with self-guiding procedures for the
downloading, merging, post-processing, and graphical/statistical analysis
of data.

<ACCESSION NO.> [illegible]
<DATABASE SOURCE> DIALOG NTIS FILE 6
<REPORT NO.> <NTIS> DE84013210
<TITLE> Online Directory of Databases for Material Properties
<AUTHORS> Hampel, V. E. ; Bollinger, W. A. ; Gaynor, C. A. ; Oldani, J.
J.
<PUB DESC> Lawrence Livermore National Lab., CA. ; <Code> 068147000; 9513035 ; Department
DC. ; UCRL-90276; CONF-8406139-1
<DATE> May 1984
<PG> 122p
<AV> Portions are illegible in microfiche products.
<LANGUAGE> English
<DOCUMENT TYPE> Conference proceeding
<PC> PC A06/MF A01
<JA> GFAI8423; NSA0900
<CO OF PUBL> United States
<NT> International CODATA conference, Jerusalem, Israel, 24 Jun 1984.
<CN> W-7405-ENG-48
<ABSTRACT> We have created an online directory of databases of material
properties on the Technology Information System at Lawrence Livermore
National Laboratory (LLNL/TIS). This directory is intended to provide
interactive access to scientific and technical databases available to
the public that contain information pertaining to nuclear, atomic, molecular,
physical, chemical, and mechanical properties of substances. The directory
is based on work done earlier by Joseph Hilsenrath of the National Bureau
of Standards (NBS/OSRD) and Jack H Westbrook of General Electric Corporation.
In addition to the 101 data files previously reported, we have updated
the information and identified more than 38 new numeric databases and
predictive systems in these fields. We have included, where applicable,
entries contained in the directories published by Cuadra Associates,
CODATA, and UNESCO. In addition to describing the contents of the databases,
we have provided updated information on the availability of the databases
and their online access over public telephone and data networks. The
online directory is prepared for use by scientists and engineers and
should enhance the sharing of S and T resources over communication networks.
This directory is expected to become particularly important to the national
and international magnetic- and laser-energy fusion projects, nuclear
criticality safety, and computer aided engineering programs. Some of
the numeric databases are directly accessible by authorized users via

describes the applications which have benefited from ARPANET during the reporting period. Finally, it discusses an investigation of the techniques for facsimile transmission between different devices over the network. Earlier work in attacking hosts by front-end techniques has been broadened to provide ** gateway ** facilities between computer networks. Here, pursued were two lines. An interne twork Transmission Control Protocol TCP has been implemented which is designed to be applicable to a host-host protocol between hosts on different networks. Experiments to test the properties of this protocol have been started between UCL, Stanford U and Bolt, Beranek and Newman BBN. More effort has been put into designing and implementing ** gateway ** functions when a specific node acts as a ** gateway ** between two networks and performs a mapping between the standard protocols of each. Investigated was the applicability of this approach to several networks, including the connection oa ARPANET and the UK Post Office Experimental Pocket Switched Service EPSS. Preliminary results show that the technique should be feasible, but since the other networks are not yet operational, the technique was not demonstrated.
<ACCESSION NO.> 77-1027
<DATABASE SOURCE> SDC/Library and Info
<TRANSLATION DATE> Mon Jul 1 13:40:50 PDT 1985 (489098450)
<DOWNLOAD DATE> Mon Jul 1 10:50:40 PDT 1985 (489088240)
<DOWNLOAD FILE NAME> sdcgate
<DATE> 1976
<TITLE> The reference department: gateway to the National Library
<AUTHORS> Umo, M.G.
<PUB DESC> Nigerbiblios, 1 (1) Jan 1976, 19-20, 22
<CO OF PUBL> English
<Category Code> RuNju
<DESCRIPTORS> Reference Work: Departments; National libraries; National Library of Nigeria; Reference Department
<Supplementary terms> Reference departments; Nigeria
<ABSTRACT> Outlines the basic responsibilities of the reference department, which offers a 12 hour a day service to users. The spread of material on various floors and the constant shifting around of stock pose problems. Briefly describes such routine tasks as: maintaining the public catalogue; shelf-reading; stocktoking; compiling the picture file of important events; and maintaining the map file. Reference desk duties are enumerated. A policy of maximum courtesy and minimum delay is adopted in attending to all enquires.
<ACCESSION NO.> A032248
<DATABASE SOURCE> DTIC/drols-tr
<TRANSLATION DATE> Mon Jul 1 13:33:43 PDT 1985 (489098023)
<DOWNLOAD DATE> Mon Jul 1 10:18:29 PDT 1985 (489086309)
<DOWNLOAD FILE NAME> gate
<FIELDS AND GROUPS> 15/5, 5/11
<ENTRY CLASSIFICATION> UNCLASSIFIED
<CORPORATE AUTHOR> RAND CORP SANTA MONICA CALIF
<TITLE> GETTING PEOPLE TO PARKS,
<TITLE CLASSIFICATION> UNCLASSIFIED
<AUTHORS> VAUGHAN,ROGER J. ;
<DATE> APR , 1976
<PAGINATION> 25P
<REPORT NUMBEF> P-5654
<REPORT CLASSIFICATION> UNCLASSIFIED
<DESCRIPTORS> *TRANSPORTATION; *PASSENGERS; *RECREATION; NEW YORK CITY(NEW YORK); NEW JERSEY; PASSENGER VEHICLES; PARKING FACILITIES; ACCESS; ECONOMIC ANALYSIS
<DESCRIPTOR CLASSIFICATION> UNCLASSIFIED
<IDENTIFIERS> *GATEWAY NATIONAL RECREATION AREA
<IDENTIFIER CLASSIFICATION> UNCLASSIFIED
<ABSTRACT> THE PURPOSE OF THIS PAPER IS TO PROVIDE AN ECONOMIC PERSPECTIVE ON THE PROBLEM OF TRANSPORTING PEOPLE TO GATEWAY NATIONAL RECREATION AREA LOCATED IN NEW YORK CITY AND NORTHEASTERN NEW JERSEY. WHILE IT DOES NOT CONTAIN ANY DETAILED EMPIRICAL CALCULATIONS FOR THE SOLUTION TO THIS COMPLEX ISSUE, IT IS HOPED THAT SOME OF THE SUGGESTIONS MIGHT BE USEFUL INPUT INTO THE PLANNING PROCESS, AND MIGHT OPEN THE WAY TO MORE DETAILED RESEARCH AND ANALYSIS.
<ABSTRACT CLASSIFICATION> UNCLASSIFIED
<INITIAL INVENTORY> 2
<LIMITATION CODES> 1
<SOURCE CODE> 296600
<DOCUMENT LOCATION> NTIS
<GEOPOLITICAL CODE> 0628
<TYPE CODE> W
<ACCESSION NO.> 78-350
<DATABASE SOURCE> SDC/Library and Info
<TRANSLATION DATE> Mon Jul 1 13:40:50 PDT 1985 (489098450)
<DOWNLOAD DATE> Mon Jul 1 10:50:40 PDT 1985 (489088240)

END

DTIC

6-86