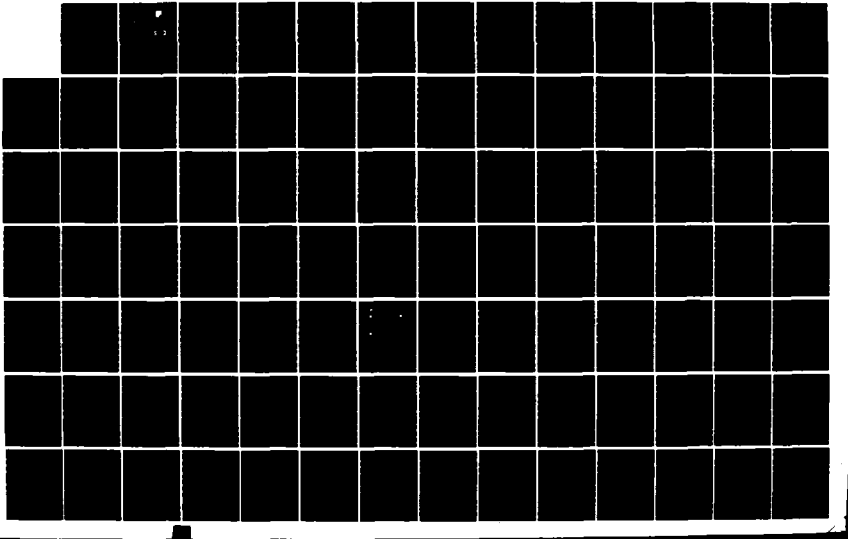
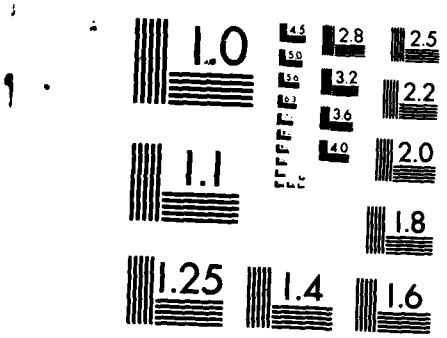
AD-A173 344 DESIGN CONCEPTS FOR DATA BASE UTILITIES(U) AOB SYSTEMS CORP HARVARD MA APR 86 RADC-TR-86-48 F30602-84-C-0011 1/3

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A173 344

RADC-TR-86-48
Final Technical Report
April 1986

DESIGN CONCEPTS FOR DATA BASE UTILITIES

AOG Systems Corporation

DTIC
ELECTE
OCT 2 2 1986
S D
D

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC FILE COPY

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

398340 SH

86 1

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-86-48 has been reviewed and is approved for publication.

APPROVED: Patrick K. McCabe

PATRICK K. McCABE
Project Engineer

APPROVED: Thaddeus J. Domurat

THADEUS J. DOMURAT
Acting Technical Director
Intelligence & Reconnaissance Division

FOR THE COMMANDER: Richard W. Pouliot

RICHARD W. POULIOT
Plans & Programs Division

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRDA) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

ADA173344

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A .	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-86-48
6a. NAME OF PERFORMING ORGANIZATION AOG Systems Corporation 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (IRDA)
6c. ADDRESS (City, State, and ZIP Code) P O Box M Harvard MA 01451	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) (IRDA)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-84-C-0011
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62702F; PROJECT NO. 4594; TASK NO 16; WORK UNIT ACCESSION NO D1

11. TITLE (Include Security Classification)
DESIGN CONCEPTS FOR DATA BASE UTILITIES

12. PERSONAL AUTHOR(S)

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Final	FROM Dec 83 TO Dec 85	April 1986	442

16. SUPPLEMENTARY NOTATION
N/A

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP
15	04	
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Data Dictionary
Long Haul Networks
Data Base Networks

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

This report describes a network resident, data access utility. This utility, based on data dictionary technology, provides user transparent access to data accessible via a network. The location, organization and structure of the data are immaterial to the user. Heterogeneous data base systems in a long haul network constitute the evaluated environment.

A support development facility for the network is also discussed.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Patrick McCabe	22b. TELEPHONE (Include Area Code) (315) 330-3126 22c. OFFICE SYMBOL RADC (IRDA)

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

DESIGN CONCEPTS FOR

DATABASE UTILITIES

TABLE OF CONTENTS

		Page
1.	**MANAGEMENT OVERVIEW**	1
1.1	Introduction	1
1.1.1	Statement of the Problem	1
1.1.2	Goals and Objectives of the IDN	2
1.1.3	Overview of the Proposed Solution	4
1.1.4	Organization of the Management Overview	5
1.2	Design Assumptions	11
1.3	Design Issues and Considerations	13
1.3.1	Security	13
1.3.2	Delegated Production Policy	14
1.3.3	The FIPS/ANSI Information Resource Dictionary System	15
1.3.4	Name Spaces	16
1.3.5	Network and Database Metadata	17
1.3.6	Query Control Blocks and Query Execution Graphs	18
1.3.7	Stored and ad hoc Queries	20
1.3.8	Control Procedures and Metadata Synchronization	20
1.3.9	Underlying Network Protocol Dependencies	22
1.4	The Proposed Solution	27
1.4.1	Query Processing	27
1.4.1.1	Subsystem 1. PROVIDE-LOCAL-DD-SUPPORT	29
1.4.1.2	Subsystem 2. VALIDATE-TOKENIZED-QUERY	32
1.4.1.3	Subsystem 3. VALIDATE-INTERMEDIATE-QUERY	33
1.4.1.4	Subsystem 4. DETERMINE-DATA-ACCESSIBILITY	34
1.4.1.5	Subsystem 5. DECOMPOSE-INTO-CANONICAL-SUBQUERIES	35
1.4.1.6	Subsystem 6. IMPLEMENT-EXECUTION-STRATEGY	35
1.4.1.7	Subsystem 7. QUERY-MONITORING-AND-CONTROL	36
1.4.1.8	Subsystem 8. RECEIVE-SUBQUERIES-AT-DESTINATION	38
1.4.1.9	Subsystem 9. TRANSLATE-TO-NETWORK-DATA-REQUEST-LANGUAGE (NDRL)	38

QUALITY INSPECTED 4

A-1

1.4.1.10 Subsystem 10. EXTRACT-AND-COMPUTE (Execute the NDRL) 39
1.4.1.11 Subsystem 11. PREPARE-OUTPUT-FOR-NETWORK .. 39
1.4.1.12 Subsystem 12. IMPLEMENT-COMPOSITION-STRATEGY 39
1.4.1.13 Subsystem 13. ASSEMBLE-COMPOSITE-RESPONSE . 40
1.4.1.14 Subsystem 14. DELIVER-COMPOSITE-RESPONSE .. 40

1.4.2 The IDN Architecture 41
1.4.2.1 The IDN Processors 41
1.4.2.2 User Node Processing 45
1.4.2.3 D1 Node Processing 47
1.4.2.4 D2Q Node Processing 47
1.4.2.5 D1* Node Processing 49
1.4.2.6 D3Q Node Processing 49
1.4.2.7 Data Node Processing 50

1.5 Summary and Conclusions 53

2. SUPPORTING TECHNICAL PAPERS 55

2.1 Response Quotas in the IDN 57

2.1.1 Multiple Thresholds of User Quotas 57
2.1.2 Establishing User Quotas Based Upon Precedence 58
2.1.2.1 Special Processing for Urgent Queries 60
2.1.2.2 Purging Queues Based on Precedence 60
2.1.2.3 Dispatching Based on Precedence 61

2.2 Query Processing Restart and Recovery 63

2.2.1 Introduction 63
2.2.2 A Review of Query Processing 65
2.2.3 Definition of a Query Failure with Gross Recovery Strategy 68
2.2.4 The Recovery Scheme 71
2.2.5 Summary 77

2.3 Subquery Processing Against the Basic Data Models 79

2.3.1 The Phases of Query Translation and the NDRL 79
2.3.2 Subquery Processing Against the Hierarchical Data Model 85
2.3.3 Subquery Processing Against the Network Data Model .. 89
2.3.4 Subquery Processing Against the Relational Data Model 94
2.3.5 Subquery Processing Against the Inverted Model 98

2.4 Using Joined Subsets (SEMI-JOINs) in Subresponse Composition 101

3. SUBSYSTEMS SPECIFICATION 105

3.1 SUBSYSTEM 1: PROVIDE-LOCAL-DD-SUPPORT 107

3.1.1 Overview 107

3.1.2 Interface Specifications 108
3.1.2.1 Intersubsystem Data Flows 108
3.1.2.2 Subsystem Events 108

3.1.3 Technical Requirements 108
3.1.3.1 Actions 108
3.1.3.2 Intrasubsystem Data Flows 110
3.1.3.3 Tasking 113
3.1.3.4 Exception Conditions 113
3.1.3.5 Rationale for Critical Decisions 113
3.1.3.6 Open / Deferred Decisions 114
3.1.3.7 Critical Algorithms 115

3.1.4 Data Requirements 115
3.1.4.1 Explanation of Major Data Flows 115
3.1.4.2 Special Data Requirements 119

3.1.5 Database Requirements 119
3.1.5.1 Network Metadata 119
3.1.5.2 Data Metadata 119
3.1.5.3 Other 121

3.1.6 Notes 121

3.2 SUBSYSTEM 2: VALIDATE-TOKENIZED-QUERY 123

3.2.1 Overview 123

3.2.2 Interface Specifications 124
3.2.2.1 Intersubsystem Data Flow Diagram 124
3.2.2.2 Subsystem Events 124

3.2.3 Technical Requirements 124
3.2.3.1 Actions 124
3.2.3.2 Intrasubsystem Data Flows 127
3.2.3.3 Tasking 130
3.2.3.4 Exception Conditions 130
3.2.3.5 Rationale for Critical Decisions 130
3.2.3.6 Open / Deferred Decisions 132
3.2.3.7 Critical Algorithms 133

3.2.4 Data Requirements 135
3.2.4.1 Explanation of Major Data Flows 135
3.2.4.2 Special Data Requirements 138

3.2.5 Database Requirements 139
3.2.5.1 Network Metadata 139
3.2.5.2 Data Metadata 139
3.2.5.3 Other 139

3.2.6 Notes 139

3.3 SUBSYSTEM 3: VALIDATE-INTERMEDIATE-QUERY 143

3.3.1 Overview 143

3.3.2 Interface Specifications 144
3.3.2.1 Intersubsystem Data Flow Diagram 144
3.3.2.2 Subsystem Events 144

3.3.3 Technical Requirements 144
3.3.3.1 Actions 144
3.3.3.2 Intrasubsystem Data Flows 144
3.3.3.3 Tasking 148
3.3.3.4 Exception Conditions 148
3.3.3.5 Rationale for Critical Decisions 148
3.3.3.6 Open / Deferred Decisions 149
3.3.3.7 Critical Algorithms 151

3.3.4 Data Requirements 152
3.3.4.1 Explanation of Major Data Flows 152
3.3.4.2 Special Data Requirements 155

3.3.5 Database Requirements 156
3.3.5.1 Network Metadata 156
3.3.5.2 Data Metadata 156
3.3.5.3 Other 159

3.3.6 Notes 159

3.4 SUBSYSTEM 4: DETERMINE-DATA-ACCESSIBILITY 161

3.4.1 Overview 161

3.4.2 Interface Specifications 161
3.4.2.1 Intersubsystem Data Flow Diagram 161
3.4.2.2 Subsystem Events 161

3.4.3 Technical Requirements 161
3.4.3.1 Actions 161
3.4.3.2 Intrasubsystem Data Flows 166
3.4.3.3 Tasking 166
3.4.3.4 Exception Conditions 168
3.4.3.5 Rationale for Critical Decisions 168
3.4.3.6 Open / Deferred Decisions 169
3.4.3.7 Critical Algorithms 170

3.4.4 Data Requirements 172
3.4.4.1 Explanation of Major Data Flows 172
3.4.4.2 Special Data Requirements 173

3.4.5 Database Requirements 173
3.4.5.1 Network Metadata 173
3.4.5.2 Data Metadata 174
3.4.5.3 Other 174

3.4.6 Notes 174

3.5 SUBSYSTEM 5: DECOMPOSE-QUERY-INTO-CANONICAL-SUBQUERIES 175

3.5.1 Overview 175

3.5.2 Interface Specifications 176
3.5.2.1 Intersubsystem Data Flow Diagram 176
3.5.2.2 Subsystem Events 176

3.5.3 Technical Requirements 176
3.5.3.1 Actions 176
3.5.3.2 Intrasubsystem Data Flows 183
3.5.3.3 Tasking 184
3.5.3.4 Exception Conditions 184
3.5.3.5 Rationale for Critical Decisions 184
3.5.3.6 Open / Deferred Decisions 185
3.5.3.7 Critical Algorithms 185

3.5.4 Data Requirements 189
3.5.4.1 Explanation of Major Data Flows 189
3.5.4.2 Special Data Requirements 189

3.5.5 Database Requirements 189
3.5.5.1 Network Metadata 189
3.5.5.2 Data Metadata 190
3.5.5.3 Other 190

3.5.6 Notes 190

3.6 SUBSYSTEM 6: IMPLEMENT-EXECUTION-STRATEGY 197

3.6.1 Overview 197

3.6.2 Interface Specifications 198
3.6.2.1 Intersubsystem Data Flow Diagram 198
3.6.2.2 Subsystem Events 198

3.6.3 Technical Requirements 198
3.6.3.1 Actions 198
3.6.3.2 Intrasubsystem Data Flows 201
3.6.3.3 Tasking 201
3.6.3.4 Exception Conditions 204
3.6.3.5 Rationale for Critical Decisions 204
3.6.3.6 Open / Deferred Decisions 204
3.6.3.7 Critical Algorithms 204

3.6.4 Data Requirements 205
3.6.4.1 Explanation of Major Data Flows 205
3.6.4.2 Special Data Requirements 207

3.6.5 Database Requirements 207
3.6.5.1 Network Metadata 207
3.6.5.2 Data Metadata 207
3.6.5.3 Other 207

3.6.6 Notes 207

3.7 SUBSYSTEM 7: QUERY-MONITORING-AND-CONTROL 209

3.7.1 Overview 209

3.7.2 Interface Specifications 211
3.7.2.1 Intersubsystem Data Flows 211
3.7.2.2 Subsystem Events 216

3.7.3 Technical Requirements 220
3.7.3.1 Actions 220
3.7.3.1.1 D1 QMC Processing 220
3.7.3.1.2 D2Q/C QMC Processing of the Query and Subqueries 221
3.7.3.1.3 D1* QMC Processing of the Subqueries 222
3.7.3.1.4 D1* and D2Q/C QMC Processing of Subresponses and Responses 222
3.7.3.1.5 QMC Purge of the Query from the IDN 224
3.7.3.2 Intrasubsystem Data Flows 225
3.7.3.3 Tasking 225
3.7.3.4 Exception Conditions 225
3.7.3.5 Rationale for Critical Decisions 227
3.7.3.6 Open / Deferred Decisions 227
3.7.3.7 Critical Algorithms 227

3.7.4 Data Requirements 228
3.7.4.1 Major Data Flows 228
3.7.4.2 Special Data Requirements 231

3.7.5 Database Requirements 231
3.7.5.1 Network Metadata 231
3.7.5.2 Data Metadata 231
3.7.5.3 Other 232

3.7.6 Notes 232

3.8 SUBSYSTEM 8: RECEIVE-SUBQUERIES 233

3.8.1 Overview 233

3.8.2 Interface Specifications 233
3.8.2.1 Intersubsystem Data Flows 233
3.8.2.2 Subsystem Events 235

3.8.3 Technical Requirements 235
3.8.3.1 Actions 235
3.8.3.2 Intrasubsystem Data Flows 237
3.8.3.3 Tasking 238
3.8.3.4 Exception Conditions 238
3.8.3.5 Rationale for Critical Decisions 239
3.8.3.6 Open / Deferred Decisions 239
3.8.3.7 Critical Algorithms 239

3.8.4 Data Requirements 239
3.8.4.1 Major Data Flows 239
3.8.4.2 Special Data Requirements 240

3.8.5 Database Requirements 240
3.8.5.1 Network Metadata 240
3.8.5.2 Data Metadata 240
3.8.5.3 Other 240

3.8.6 Notes 240

3.9 SUBSYSTEM 9: TRANSLATE-TO-NETWORK-DATA-REQUEST-LANGUAGE ... 243

3.9.1 Overview 243

3.9.2 Interface Specifications 244
3.9.2.1 Intersubsystem Data Flows 244
3.9.2.2 Subsystem Events 244

3.9.3 Technical Requirements 247
3.9.3.1 Actions 247
3.9.3.2 Intrasubsystem Data Flows 252
3.9.3.3 Tasking 252
3.9.3.4 Exception Conditions 252
3.9.3.5 Rationale for Critical Decisions 254
3.9.3.6 Open / Deferred Decisions 255
3.9.3.7 Critical Algorithms 255

3.9.4 Data Requirements 256
3.9.4.1 Major Data Flows 256
3.9.4.2 Special Data Requirements 256

3.9.5 Database Requirements 257
3.9.5.1 Network Metadata 257
3.9.5.2 Data Metadata 257
3.9.5.3 Other 258

3.9.6 Notes 258

3.10 SUBSYSTEM 10: EXTRACT-AND-COMPUTE-DATA 259

3.10.1 Overview 259

3.10.2 Interface Specifications 260
3.10.2.1 Intersubsystem Data Flows 260
3.10.2.2 Subsystem Events 260

3.10.3 Technical Requirements 260
3.10.3.1 Actions 260
3.10.3.2 Intrasubsystem Data Flows 264
3.10.3.3 Tasking 264
3.10.3.4 Exception Conditions 266
3.10.3.5 Rationale for Critical Decisions 266
3.10.3.6 Open / Deferred Decisions 268
3.10.3.7 Critical Algorithms 268

3.10.4 Data Requirements 269
3.10.4.1 Explanation of Major Data Flows 269
3.10.4.2 Special Data Requirements 271

3.10.5 Database Requirements 271
3.10.5.1 Network Metadata 271
3.10.5.2 Data Metadata 271
3.10.5.3 Other 272

3.10.6 Notes 272

3.11 SUBSYSTEM 11: PREPARE-OUTPUT-FOR-NETWORK 273

3.11.1 Overview 273

3.11.2 Interface Specifications 274
3.11.2.1 Intersubsystem Data Flows 274
3.11.2.2 Subsystem Events 274

3.11.3 Technical Requirements 274
3.11.3.1 Actions 274
3.11.3.2 Intrasubsystem Data Flows 277
3.11.3.3 Tasking 277
3.11.3.4 Exception Conditions 277
3.11.3.5 Rationale for Critical Decisions 279
3.11.3.6 Open / Deferred Decisions 279
3.11.3.7 Critical Algorithms 280

3.11.4 Data Requirements 280
3.11.4.1 Explanation of Major Data Flows 280
3.11.4.2 Special Data Requirements 282

3.11.5 Database Requirements 283
3.11.5.1 Network Metadata 283
3.11.5.2 Data Metadata 283
3.11.5.3 Other 283

3.11.6 Notes 283

3.12 SUBSYSTEM 12: IMPLEMENT-COMPOSITION-STRATEGY 285

3.12.1 Overview 285

3.12.2 Interface Specifications 286
3.12.2.1 Intersubsystem Data Flows 286
3.12.2.2 Subsystem Events 286

3.12.3 Technical Requirements 289
3.12.3.1 Actions 289
3.12.3.2 Intrasubsystem Data Flows 291
3.12.3.3 Tasking 291
3.12.3.4 Exception Conditions 293
3.12.3.5 Rationale for Critical Decisions 293
3.12.3.6 Open / Deferred Decisions 293
3.12.3.7 Critical Algorithms 293

3.12.4 Data Requirements 294
3.12.4.1 Major Data Flows 294
3.12.4.2 Special Data Requirements 294

3.12.5 Database Requirements 295
3.12.5.1 Network Metadata 295
3.12.5.2 Data Metadata 295
3.12.5.3 Other 295

3.12.6 Notes 295

3.13 SUBSYSTEM 13: ASSEMBLE-COMPOSITE-RESPONSE 297

3.13.1 Overview 297

3.13.2 Interface Specifications 298
3.13.2.1 Intersubsystem Data Flow Diagram 298
3.13.2.2 Subsystem Events 298

3.13.3 Technical Requirements 301
3.13.3.1 Actions 301
3.13.3.2 Intrasubsystem Data Flows 303
3.13.3.3 Tasking 303
3.13.3.4 Exception Conditions 303
3.13.3.5 Rationale for Critical Decisions 303
3.13.3.6 Open / Deferred Decisions 305
3.13.3.7 Critical Algorithms 305

3.13.4 Data Requirements 305
3.13.4.1 Major Data Flows 305
3.13.4.2 Special Data Requirements 306

3.13.5 Database Requirements 306
3.13.5.1 Network Metadata 306
3.13.5.2 Data Metadata 306
3.13.5.3 Other 306

3.13.6 Notes 306

3.14 SUBSYSTEM 14: DELIVER-COMPOSITE-RESPONSE 307

3.14.1 Overview 307

3.14.2 Interface Specifications 308
3.14.2.1 Intersubsystem Data Flows 308
3.14.2.2 Subsystem Events 308

3.14.3 Technical Requirements 308
3.14.3.1 Actions 308
3.14.3.2 Intrasubsystem Data Flows 312
3.14.3.3 Tasking 312
3.14.3.4 Exception Conditions 314
3.14.3.5 Rationale for Critical Decisions 314
3.14.3.6 Open / Deferred Decisions 315
3.14.3.7 Critical Algorithms 315

3.14.4 Data Requirements 315
3.14.4.1 Major Data Flows 315
3.14.4.2 Special Data Requirements 315

3.14.5 Database Requirements 316
3.14.5.1 Network Metadata 316
3.14.5.2 Data Metadata 316
3.14.5.3 Other 316

3.14.6 Notes 316

4. IDN MANAGEMENT UTILITIES 317

4.1 Introduction 317

4.1.1 Approach 317
4.1.2 Definitions 317
4.1.3 User Communities 318

4.2 Utility Requirements 320

4.2.1 IDN Support Group Requirements 320
4.2.1.1 Software Development and Maintenance Requirements 321
4.2.1.1.1 Initial Development 321
4.2.1.1.2 Ongoing Development and Maintenance Requirements 322
4.2.1.2 Global Metadata Management and Distribution Control 323

4.2.2 IDN Operations Management Requirements 324
4.2.2.1 The D1 Processor 325
4.2.2.2 The D2Q Processor 327
4.2.2.3 The D1* Processor 328

4.2.3 Network Administration Requirements 329

4.3 The IDNSF 330

4.3.1 The IDN Software Development Environment 332
4.3.2 The Data Analyst "Workbench" 334
4.3.3 The IDN Configuration Management System 337
4.3.4 The IDN Distribution & Distribution Control System .. 339
4.3.5 The IDN Simulation 340
4.3.6 The IDN Service Request Reporting System 340
4.3.7 The IDN Support Group Activity Reporting System 341

4.4 Operational Support Utilities 341

4.4.1 Node Operation Utilities 342

4.4.2 Query-User Support Utilities 343

4.4.3 Monitoring and Analysis Utilities 345

4.4.4 Modeling and Design Aids 347

4.4.5 Change Implementation 349
4.4.5.1 Database Changes 350
4.4.5.1.1 Switching Physical Databases 350
4.4.5.1.2 Switching Both Logical and Physical Databases 351
4.4.5.2 Changing the Network 354
4.4.5.3 Software Installation 356
4.4.5.4 The General Approach to Change Implementation 356

4.4.6 IDNSF Interface 357
4.4.6.1 Network Reconfiguration 358
4.4.6.2 Implementing Logical Database Changes 359
4.4.6.3 Implementing IDN Software Changes 359
4.4.6.4 Resolving Operational Problems 360
4.4.6.5 Requesting Software Enhancements 362
4.4.6.6 Requesting Technical Assistance 362

4.4.7 Procedures 363

4.5 Issues 365

4.5.1 IDNSF Issues 365
4.5.2 Operational Support Utilities Issues 366

APPENDICES

APPENDIX I. DETAILED DESCRIPTIONS OF MAJOR INTER-SUBSYSTEM DATA FLOWS 371

0. Index 372
1. User-Node to D1 Data Flow Definitions 373
2. D1 to D1 Data Flow Definitions 376
3. D1 to D2Q Data Flow Definitions 377
4. D2Q to D2Q Data Flow Definitions 378
5. D2Q to D1* Data Flow Definitions 379
6. D1* to Data Node Data Flow Definitions 380
7. Data Node to D1* Data Flow Definitions 381
8. D1* to D2Q Data Flow Definitions 382
9. D1* to D1* Data Flow Definitions 384
10. D2Q to D1 Data Flow Definitions 385
11. D1*-to-D1 Data Flow Definitions 386
12. D1 to User-Node Data Flow Definitions 388
13. Tokenized Query 389
14. Intermediate Query 393
15. Query Syntax - Common Data Items 398
16. Query Control Block 399
17. Common Data Items 404

APPENDIX II. IDN IRDS SCHEMA 407

1. Introduction 407

2. Entity-Types 408
   2.1 Group 1: Security Related Entity-Types 408
   2.2 Group 2: Query Processing Related Entity-Types ... 408
   2.3 Data Definition Entity-Types 409
      2.3.1 Group 3: IDN Global Data Definition Entity-Types 409
      2.3.2 Groups 4 - 7: Target Database Definition .. 410
      2.3.3 Group 8: Basic Data Type Entity-Types 411

3. Relationship-Types 412
   3.1 Relationship-types Supporting the IDN Query User .. 412
   3.2 Relationship-types Defining CDRs 412
   3.3 Relationship-types Defining Target Database Views . 413
   3.4 Relationship-types Defining the Correspondence to Target Databases 414
   3.5 Relationship-types Defining the Static Network View 415

4. Attribute-Types 416

5. Entity-Type / Attribute-Type Correspondence 418

6. Relationship-Type / Attribute-Type Correspondence 420

7. Attribute-Group-Types 422

8. IRDS Dictionary-Views 423

CHAPTER 1

MANAGEMENT OVERVIEW

1.1 INTRODUCTION

This chapter presents a management overview of the functional description of the Integrated Data Network (IDN). It is intended to provide a brief analysis of IDN goals, requirements and issues, along with a high-level description of the proposed IDN design.

1.1.1 STATEMENT OF THE PROBLEM

The Integrated Data Network provides the necessary facilities for the management and control of the processing of queries in a network of heterogeneous Database Management Systems. This network consists of the following components:

- User Nodes with standard network interfaces.
- Data Nodes with various Database Management Systems.
- An interconnecting communications network.

Thus, the IDN architecture supports a heterogeneous collection of User Nodes and Data Nodes and the processing of queries between them. The User Nodes, which range from individual terminals to host mainframes, send queries to and receive responses from the IDN. These queries are directed to one or more databases managed by the Data Nodes, which may be host mainframes or even database machines. The structure(s) of the databases and the nature of the Database Management Systems (DBMSs)

which maintain them may vary widely. Particular data record types, data elements and their respective values may be unique to a particular database and Data Node, or may be distributed across several Data Nodes.

The basic objective of the IDN is that it must support a heterogeneous network with redundant data which may be structured differently at each of the Data Nodes, and that the following fundamental conditions must be met:

- All user interaction must occur through a common interface.
- A query may be satisfied using data from multiple Data Nodes.
- The database structures and Database Manipulation Languages or Data Query Languages at each Data Node may be different, and are not known to the user.
- The user, in general, will not know the location of the data required to respond to a query.

1.1.2 GOALS AND OBJECTIVES OF THE IDN

Stated goals for the Integrated Data Network design generally fall into six categories:

1. **Control** - over query processing and the network metadata (including metadata protection).
2. **Reliability** - through simplicity of design and minimal impact on the IDN as a whole.
3. **Survivability** - via interchangeability and redundancy.
4. **Timeliness of Responses** - by virtue of efficient communications and load balancing.

5. **Acceptable Operational Cost** - in terms of communications costs, processing costs, and the balancing of the network load.

6. **Acceptable Impact** - on existing facilities including User Nodes, Data Nodes, and network switching and routing systems.

Consideration of these goals and the initial analysis of the problem resulted in the derivation of two additional design objectives for the IDN. These are:

- **Minimization of New Software** - which must be developed in support of the IDN. Meeting this objective results in:
  - Increased reliability of the system since the software can be exercised more frequently and more thoroughly during the development cycle, thereby permitting bugs to be discovered sooner.
  - Increased reliability since less software must be transported and set up at alternate sites.
  - Reduced cost of software development and the operational cost of query processor maintenance.
- **Homogeneity of New Software** - and the environment in which it operates. Meeting this objective results in:
  - Increased reliability of the software, as there is only a single version of the software and the number of interfaces is minimized.
  - Increased survivability of the system due to greater redundancy and easier replacement.
  - Reduced maintenance and operation costs.

1.1.3 OVERVIEW OF THE PROPOSED SOLUTION

The IDN architecture design being presented features a functional hierarchy of special Dictionary/Directory/Query (D/D/Q) processors, the "IDN Nodes", for performing the query processing functions and maintaining the metadata required. This design has evolved from the development and analysis of a number of alternative architecture scenarios. The design takes into account both the goals established and the general assumptions made for the IDN. The approach was especially influenced by the desire to minimize the quantity of new software which might have to be developed to implement the IDN query processing capability, and to make that software as homogeneous as possible.

The IDN Processors are supported by a common Data Dictionary/Directory System (DD/DS) which is based on the proposed Federal Information Processing Standard (FIPS) and American National Standard (ANS) for Information Resource Dictionary Systems; this Standard is discussed more fully in Section 1.3.3. It should be noted that the term "Information Resource Dictionary System" (IRDS) will be used in place of "Data Dictionary/Directory System" throughout this document, and the term "dictionary" in place of "Data Dictionary/Directory". A common IRDS schema, i.e., the definition of the structure of the dictionaries, is shared by those processors which have the same roles and are of the same type. Commonalities are maintained between the schemas of IDN Processors of different types. The IRDS schemas describe the static and dynamic reference information needed for processing a query from any User Node against any Data Nodes.

The IDN design provides solutions to problems such as how to translate a query into an internal form, how to map a query to one or more target databases, executing subqueries against those databases, and delivering a composite response back to the user. To meet these tasks effectively and efficiently, the IDN provides facilities for:

- Translation of the user query into an IDN-standard intermediate format.
- Validation of the query.
- Decomposition of the query into "subqueries" when the data required to satisfy the query occurs at multiple Data Nodes.
- Translation of each subquery into a form which can be mapped into the language of the target database system(s).
- Composition of the results from multiple Data Nodes into a single response which may be sent back to the user.

It is obvious that the process leading to the specification of this design required the analysis of a wide range of design tradeoffs and alternatives. While these tradeoffs and designs will not be discussed in detail within this Management Overview, Table 1.1 provides a summary of the tradeoff areas analyzed, the alternatives considered, and the recommended alternative.

1.1.4 ORGANIZATION OF THE MANAGEMENT OVERVIEW

Before a full understanding the proposed solution for the IDN problem may be achieved, it is necessary to present the underlying assumptions upon which the design was based. These assumptions are found in Section 1.2 of this document.

Design issues and considerations which affected the eventual IDN design are presented in Section 1.3.

Once the set of design requirements and considerations have been presented, an overview of the proposed IDN solution will be provided; this is done in Section 1.4, and it includes a discussion of the steps

required to process a query from the initial input to the final response, and a description of the IDN architecture. The IDN architecture is designed to support the entire query processing scenario and to provide a basis for addressing all design requirements, assumptions and considerations.

TRADEOFF AREA	ALTERNATIVES
Hardware vs. Software Solution to Decentralized Database Problem.	o Implement IDN in Software only. * Implement IDN in with dedicated processors and software.
Single vs. Multiple Data Dictionary / Directories.	o Single global DD/D with all network and database metadata. * DD/Ds with metadata appropriate to query processing function.
Single vs. Multiple IDN Processor Types.	o One processor type for all query processing functions. * Multiple processor types differentiated by functions and metadata.
Central vs. Distributed Control of Query Processing.	o Control through "control nodes". o Control through "control blocks". * Hybrid (node / block) control mechanism.
Survivability vs. Metadata Maintenance Overhead.	o Minimum metadata at each IDN Node. * Redundant metadata at IDN Nodes.
Control vs. Communications and Processing Costs.	o Minimum costs with minimum control. o Maximum control with increased cost. * Hybrid control scheme, optimal cost.
Traditional Network Directories vs. FIPS / ANS IRDS Extension.	o Custom, "hard-coded" directories. * Uniform, extensible IRDS derivation.
Local User Names vs. Standard User / Network Access Name.	o Standard Access Names for all Users * Local Names for each User community.

A * is used to indicate the recommended alternative. In some tradeoff areas insufficient information is currently available to allow for a recommendation.

TABLE 1.1 - Page 1 Of 4

TRADEOFF AREAS	ALTERNATIVES
Standard Access Names vs. Internal IDs for Each Element.	o Standard Names used throughout network. * Internal IDs used throughout network.
Static vs. Dynamic Network Metadata (Network "Views").	o Only static view maintained. * Dynamic and static views maintained.
Single or Multiple Database Descriptions.	o Single description for all. * Individual description for each.
Transmission of Query & Sub-Query Control Blocks.	o Maintain at the generation node. * Pass along with query / subquery.
Query Archive for Reviewing / Analyzing Queries.	o Maintain on a routine basis. * Generate as needed for monitoring.
Query Execution Graph Control and Transmission.	* Maintain at the D2Q "Control Node". o Pass along with the query / subquery.
Stored vs. Ad Hoc Queries.	o Permit only Ad Hoc/original queries. * Allow query storage for future use.
One Query at a Time per User vs. Multiple Queries / User.	o Receive response before next query. * Permit queries before last response. (A limit will be set on the number of concurrent queries per user.)
Event vs. Interval Driven "Query Control Protocol".	o Event triggered control messages. o Interval triggered control messages. * Both event and interval messages.

A * is used to indicate the recommended alternative. In some tradeoff areas insufficient information is currently available to allow for a recommendation.

TABLE 1.1 - Page 2 of 4

TRADEOFF AREA	ALTERNATIVES
Abort Query Upon Timeout or Continue from Last Checkpoint.	o Require User to resubmit query. o Attempt auto-restart of query. o Interrogate User about restart. * Restart / ask User to give conditions.
Maintain Status Information as Up / Down vs. Activity Level.	o Maintain only node / link up or down. o Maintain percent busy each resource.
Semantic Interpretation of Queries, Decomposition, and Mapping to Target Databases.	o Map based upon dependencies. o Use semantic "weights" for elements. o Interrogate User about ambiguities. o Validate using data element domains. * Use all above methods appropriately.
"Non-Procedural" vs. "Procedural" User Queries.	* Permit only "non-procedural" query. o Permit both kinds of query forms.
Data Selection / Retrieval vs. Computational Queries.	o Allow only data selection / retrieval. * Allow some computations / derivations.
Failure of the "Controlling Node" for a Query.	o Restart the query from User Node. * Assign alternate "shadow nodes".
Determination of Alternate Data Sources for a Query.	* Determine at decomposition time. o Determine only when source is down.
Restart Query vs. Restart Subqueries of a Given Query.	* Restart using the validated query. o Restart using subqueries. * Restart from "shadow nodes".
Provide a Query Interface to Each Database vs. Use the Report Facilities Available.	o Provide a query interface / DBMS. o Use report facilities of DBMS. * Provide interface to report writer.

A * is used to indicate the recommended alternative. In some tradeoff areas insufficient information is currently available to allow for a recommendation.

TABLE 1.1 - Page 3 of 4

TRADEOFF AREA	ALTERNATIVES
Perform Selection and Joins. Once per DB or in Stages [JOINS and Semi-JOINS].	o Select data once and then JOIN. o Select data, JOIN, and select. * IDN determines appropriate strategy.
	(A limit will be set on the size of a sub-response.)
Place Response in User Work-Space for Further Processing.	o Save only current respose buffer. o Save multiple responses per user.
	(A limit will be set on the size of the response buffer.)
Strategy for Response Composition.	o Compose based upon response volume. o Compose based upon query semantics. * Use "intelligent" composition based upon all available information.
	(A limit will be set on the size of a response.)

A * is used to indicate the recommended alternative. In some tradeoff areas insufficient information is currently available to allow for a recommendation.

Table 1.1 - Page 4 of 4

1.2 DESIGN ASSUMPTIONS

AOG System Corporation's (AOG) analysis of the IDN requirements, and discussions with Rome Air Development Center (RADC), have determined that certain assumptions can be made regarding query processing across the network. The following assumptions apply to any IDN architecture scenario and any proposed solution:

- **Security** - The data and users of the network may be at different levels of security classification. The group of nodes participating in the resolution of a specific query operate at system high.

- **Concurrency** - It is assumed that concurrency of the data is not a concern, since database update and maintenance is outside the scope of the query processing problem.

- **Synchronization** - Synchronization of the network metadata and logical definitions of the databases in the network is assumed not to be a problem since database descriptions are relatively non-volatile.

- **Local User Interfaces** - Since each local node may provide its own custom interface to the global network query processor, it will not be necessary to modify the local user interfaces.

- **Subnets and their Protocols** - The IDN may be viewed as a single network of user nodes and data nodes operating under the TCP/IP or some other protocol; hence, the existence of specialized subnets has little or no impact.

- **Control** - Control over the processing of queries and responses is of major concern. The Delegated Production Policy (DPP) is assumed to have been extended into a set of procedures by the network administrator. Network metadata and functions must be administered and maintained, and audit trails must be generated.

- **Timeliness of Responses** - Response timeliness is an obvious concern, since the IDN is an intelligence network and queries may have high priority.

- **Appropriate Responses** - Appropriate responses are also important. Queries must be interpreted properly and the source of information used (as determined by the DPP and the user) can significantly influence the degree of confidence the user has with respect to the validity of the data.

- **Minimization of Impact** - Minimization of impact upon the existing network is a key factor. Neither the performance nor functions of existing network systems can be adversely affected.

- **Reliability** - Reliability of the IDN as a whole is a concern due not only to the importance of the query processing function, but also to the widely varying loads that may be placed upon the network.

- **Survivability** - Given the worldwide scope of the network and the need for it to function even if one or more components might fail, network survivability must be effectively addressed.

- **Operating Cost** - The cost of day-to-day network operations is to be kept at a reasonable level.

1.3 DESIGN ISSUES AND CONSIDERATIONS

The following sections present brief discussions of design issues and considerations which surfaced during AOG's design of the Integrated Data Network.

1.3.1 SECURITY

An awareness of certain aspects of network security is an important factor in the design of the IDN, and additional security considerations may further impact the ultimate system design. The aspects of network security which have been considered are:

- Each IDN User has a multilevel/multi-compartment security classification.
- Each User Node can have a multi-level security capability, if so required.
- The network will carry multi-level encrypted messages.
- A database at a Data Node is classified at the level of the highest level data element within the database.
- Queries (and resultant subqueries) are executed only if the User's security level is hierarchically greater than or equal to the classification of all the database(s) being accessed and
- The IRDSs existing at the nodes of the IDN restrict access to views of the data at the Data Nodes and work in conjunction

with the multilevel security facilities provided by the network.

The IDN can be made secure by maintaining a simple and well-defined interface between the User Nodes and the network and by maintaining all sensitive metadata and data under the control of the network. The interface to the User Nodes thus forms a security perimeter for the network and access to all classified information is controlled beginning at this perimeter.

1.3.2 DELEGATED PRODUCTION POLICY

The Delegated Production Policy (DPP) is a policy established to assign particular agencies responsibility for the production of certain intelligence information. Because the measure of confidence of the data (generally determined by its source) may be as important as the data itself, the existence of the DPP has a significant impact on the routing of queries and subqueries in the IDN.

The DPP impacts the dictionary schemas for the different types of IDN Processors. The definition of schema entities and relationships in the DPP provides a means of modeling the DPP; this metadata residing in the dictionary will control the assignment of queries and subqueries to target databases. The DPP information can be readily maintained across the IDN as part of a standard Network Data Administration procedure.

An over-ride mechanism, the "virtual key", is provided for those users who know of particular databases and have permission to direct their queries against them.

1.3.3 THE FIPS/ANSI INFORMATION RESOURCE DICTIONARY SYSTEM

The functional description of the IDN relies to a large extent on the specification of the proposed Federal Information Processing Standard (FIPS) and draft proposed American National Standard (dpANS) for Information Resource Dictionary Systems which has been produced by AOG Systems Corporation. This specification, entitled "draft proposed American National Standard IRDS", completed Public Review in October 1985.

The Standard IRDS provides an evolutionary foundation that can respond to changes in the environment as well as changes in requirements. The use of the Standard IRDS in the design of the IDN introduces a number of concepts which tend to provide integration of the network as well as uniform support of its operation. The most important of these concepts are:

- The Standard IRDS uses two layers of description: the Dictionary, which contains the metadata description of the IDN, and the dictionary schema, which contains the model of the dictionary. Both layers follow a consistent Entity-Relationship model.

- Rules can be stored in both the schema and the dictionary. These rules ensure the integrity of the metadata and govern the use of the metadata in the control of the processing of the actual information it describes.

- The Standard IRDS contains security facilities for access control to the schema and the dictionary.

- The schema can be extended to reflect changes in the information processing environment. This concept is important

in allowing the IDN to adjust to new DBMSs and possible changes in the underlying Network Protocol.

- The Standard IRDS has facilities to partition the dictionary in a variety of ways and to control the contents of each partition and the relationships between entities in different partitions.

1.3.4 NAME SPACES

There exist a number of different name spaces in the IDN, each one is used for a specific purpose. These are:

1. **The Local User Name Space (LUNS)** - a name space that the IDN user at a given node employs in formulating a query, and which is used by the system in identifying the output for a query. A number of distinct LUNSs can exist at any one node, each one corresponding to a different community of users. For security and control reasons, an object cannot be accessed by an IDN user other than through the LUNS.

2. **The Network Access Name Space (NANS)** - a global name space of unique internal identifiers for every object in the IDN.

3. **The Database Access Name Space (DANS)** - the names by which database objects are known in the databases in which they reside at Data Nodes.

There also exist another set of reserved words consisting of "computation" names (SUM, AVERAGE, etc.) and "standard" names (DATE, TIME, etc.). Each IDN processor, using the dictionary available to it, will be responsible for the translation of names from one name space to another.

1.3.5 NETWORK AND DATABASE METADATA

IDN query processing relies upon the creation and maintenance of a store of knowledge about the network and its databases. The information used to process queries may be static, as in the case of database descriptions, or dynamic, as in the case of network status. Some metadata is maintained redundantly; other metadata is partitioned. The query processing functions are assumed to have available the following network metadata:

- **Local User Name Space (LUNS)** - As defined in Section 1.3.4, user terms which map to the internal system IDs for the data elements accessible through the network for a group of users at a given node.

- **System IDs Directory** - The logical addresses (databases and Data Nodes) for each data element represented by a system ID. (The system IDs belong to are the NANS, the Network Access Name space.)

- **Database Access Name Space (DANS)** - Also defined in Section 1.3.4, local database names for each data element at a Data Node which is accessible through the network.

- **Static Network View** - A static description of the network which includes the logical types and addresses of all nodes accessible through the network and the DPP information necessary to control access to sensitive data.

- **Dynamic Network View** - Dynamic network/node status information (e.g., percent busy) needed to route and optimize queries.

- **Coherent Database Representations (CDRs)** - A concept which includes a logical description of the data elements, record-types, and keys for each database accessible through the

network. The record-types within each CDR are in at least third-normal form.

- **Database Subschemas** - These subschemas define the views of the databases available to the network.

- **Query Control Blocks (QCBs)** - A set of data maintained for each query for the duration of its processing; the set includes the source of the query, the number and destinations of subqueries and the status of the response(s).

- **Query Execution Graph (QEG)** - A scheduling and control block generated by the IDN in order to control the processing of multiple subqueries and the composition of their responses.

- **Subquery Control Blocks (SQCBs)** - A set of data maintained for each subquery for the duration of its processing; the set includes the ID of the parent query's QCB, the destination of the subquery, the destination of the subresponse, and various data translation status flags.

In addition to the above metadata, some additional metadata, needed for Network and Data Administration is accumulated over time. An example is the "Query Archive" metadata, which may be used to analyze the queries performed over some period of time. This metadata may be used, for example, to determine who issued particular queries, and to aid in tuning the network.

1.3.6 QUERY CONTROL BLOCKS AND QUERY EXECUTION GRAPHS

Query Control Blocks (QCBs) are generated for each query accepted by the network. Each QCB is identified by a unique system-generated identifier. The QCB is produced and maintained at the node which performs the query decomposition (the "controlling node" with respect to subquery execution and response composition). A copy of the QCB is then

passed along to each IDN Node which may backup the controlling node in event of that node's failure.

The metadata contained in the QCB includes:

- The query in canonical form.
- The address of the User Node submitting the request.
- The security level of the User submitting the request.
- The addresses of all target databases which satisfy the request.
- Any priority codes which may been attached to the query.
- The Query Execution Graph (QEG).
- Subquery Control Blocks (SQCBs).

The Query Execution Graph (QEG) is a partial ordering of SQCBs. The partial ordering defines the order of subquery execution. The QEG contains scheduling and control information for the execution of subqueries and the composition of responses.

When a query is decomposed into one or more subqueries a Subquery Control Block (SQCB) is generated and a copy is passed along to each IDN node subsequently involved in the processing of the subquery and its response. The metadata contained in the SQCB is similar to that contained in the QCB except that it applies to the subquery only. The SQCB includes the address of the IDN node to which responses should be dispatched.

Both the QCBs and SQCBs include variable information in addition to the fixed information described above. The result of the performance of each query processing function on the query or subquery is recorded in

all copies of the appropriate QCBs and SQCBs through event control messages passed back from the processing nodes. The failure of an expected event to be recorded within an established timeout interval results in an interval control message being sent forward to the node(s) which were to have performed the processing. A detailed explanation of control messages is provided as a separate subject in this section.

1.3.7 STORED AND AD HOC QUERIES

The IDN requires that the user be able to formulate a variety of different queries ranging from simple routine requests to more complex special queries. Any query which is likely to be resubmitted at a later time may be stored at the User Node in order to eliminate the need to compose it again. It may also prove advantageous to store some of the less common but more complex queries to serve as examples or templates when similar requests need to be made. All queries must pass through a validation process and be converted into a standard internal format before they can be stored and registered with the User Node.

To support the user, the local IRDS provides the following: help facilities; an inventory listing of available queries; the ability to create, save and recall stored queries; definitions of names used (e.g., data element names, derivation or computation names, and system defined names); and information concerning the default display format of output.

1.3.8 CONTROL PROCEDURES AND METADATA SYNCHRONIZATION

In order to support the processing of queries across the IDN, a well-conceived system of control procedures must be established. Any discussion of query processing must assume that the proper control mechanisms are in place if processing is to proceed smoothly and error conditions are to be dealt with in an orderly manner.

The IDN control system can be viewed from two complementary perspectives: Normal Query Processing Control and Exception-oriented Query Processing Control. The normal control of query processing requires that the following sequence of functions be routinely executed:

1. Receive the translated query from the User and create a Query Control Block.
2. Validate the syntax and semantics of the Query.
3. Determine the accessibility of the required data.
4. Decompose the query into canonical subqueries.
5. Generate a Query Execution Graph and Subquery Control Blocks.
6. Receive subqueries at the destination nodes.
7. Translate the subqueries into Network Data Request Language.
8. Extract and compute the response(s).
9. Prepare the output into network standard form.
10. Route and dispatch the response.
11. Compose the composite response.
12. Deliver and present the response to the user interface.

The normal control of query processing requires the sequential execution of the preceding functions. Control is maintained through the generation of Query and Subquery Control Blocks (QCBs and SQCBs) and Query Execution Graphs (QEGs).

Exceptions to normal processing flow can, of course, occur at any query processing step or between any two steps. Since the QCBs and SQCBs track the progress of the query and its subqueries, and since control messages exist to keep this information up-to-date, error conditions and exceptions can be dealt with as they arise and the system can recover gracefully from any such circumstances.

The IDN control system must generate a variety of control messages. These messages are of two principal types:

- Event driven messages.
- Interval driven messages.

Together, these two types of messages provide an "IDN Query Control Protocol" which is independent of any underlying network or transmission protocols. The event driven messages occur whenever a significant processing event has been completed. These messages are used to update the QCB and SQCB copies at the preceding nodes in the processing sequence and usually proceed "backward" along the execution path. Interval driven messages occur whenever an expected event has not occurred within an established timeout period. These messages usually proceed "forward" along the execution path from the node which last successfully completed an operation to the node(s) which are to complete the next function.

1.3.9 UNDERLYING NETWORK PROTOCOL DEPENDENCIES

It is important to consider the strong dependencies of the IDN design on support from the underlying network protocol (such as TCP/IP). The IDN must receive from the protocol:

- Status information for network nodes.
- Status information for network communications links.

- Automatic (transparent) routing of queries, subqueries, responses, and subresponses.

The status information includes whether or not a resource is available and, if so, how busy it is. Network and node status must be available to all IDN Processors, User Nodes, and Data Nodes as needed to perform query processing functions.

Communications link status information does not have to be known in detail to the IDN processors since the "transport layer" of the underlying protocol should handle routing and dispatching in a manner transparent to the user session. Link status is significant, however, if a node has become inaccessible or if major trunks are overloaded with traffic. When these conditions arise, the status information must be communicated to the higher level IDN processes (and to the Network Data Administrator) in order to optimize and control query processing.

Once a potential destination for a query, subquery, response, or subresponse has been determined to be available, the underlying protocol should handle the decomposition of messages into packets, the routing of messages, and the composition of packets back into the original messages. The query processing functions should only have to supply a destination address. All data transport functions should be completely transparent to query processing.

A summary of the network services, and the metadata required to support each of these network services, is given in Table 1.2.

NETWORK SERVICES	ISO-OSI LEVEL	METADATA REQUIRED
Data Interpretation and Display.	APPLICATION, PRESENTATION	Local-User-Name Space Network-Access-Name Space Database-Access-Name Space Coherent Database Representations
Transparent File System (Access and Transfer Services).	APPLICATION, PRESENTATION	Static Network View Dynamic Network View Coherent Database Representation Query Control Blocks Subquery Control Blocks Query Execution Graphs
Process-to-Process Communications and Host-to-Host Protocol.	SESSION	Static Network View Dynamic Network View Coherent Database Representation Query Control Blocks Subquery Control Blocks Query Execution Graphs

Table 1.2 - SUMMARY OF NETWORK SERVICES FOR THE IDN FUNCTIONAL DESCRIPTION - Page 1 of 3

NETWORK SERVICES	ISO-OSI LEVEL	METADATA REQUIRED
Session Administration - binding of communicating presentation entities.	SESSION	Static Network View Dynamic Network Views Coherent Database Representation Query Control Blocks Subquery Control Blocks Query Execution Graphs
Session Dialogue - the control of data exchange between presentation entities.	SESSION	Static Network View Dynamic Network View Coherent Database Representation Query Control Blocks Subquery Control Blocks Query Execution Graphs
Precedence, Security Classification and Compartmentation information.	TRANSPORT	Static Network View Dynamic Network View Coherent Database Representation Query Control Blocks Subquery Control Blocks Query Execution Graphs

Table 1.2 - SUMMARY OF NETWORK SERVICES FOR THE IDN FUNCTIONAL DESCRIPTION - Page 2 of 3

NETWORK SERVICES	ISO-OSI LEVEL	METADATA REQUIRED
Management of End-to-End Data Flow (data transportation)	TRANSPORT	Not Applicable
Virtual Circuit Management (in packet switching nets).	TRANSPORT	Not Applicable
Multiplexing, Flow Control, and Sequencing of Data Packets.	TRANSPORT	Not Applicable
Data Communications Error Control (detection and correction).	TRANSPORT, NETWORK	Not Applicable
Protocol Designed for Multi-Network Environments (Data Transformation).	TRANSPORT, NETWORK, DATA LINK, PHYSICAL	Not Applicable

Table 1.2 - SUMMARY OF NETWORK SERVICES FOR THE IDN FUNCTIONAL DESCRIPTION - Page 3 of 3

1.4 THE PROPOSED SOLUTION

The solution to the problem of providing a design to support the full requirements of the Integrated Data Network may be divided into two basic areas:

- A specification of the subsystems which process queries in the IDN, and
- A description of the special IDN Processors which will be used to perform the majority of the query processing activities.

In Section 1.4.1 below, a complete query processing scenario, from the initial input of the request through to the presentation of the response, is presented. Emphasis is placed on what must be done rather than how or where the action should be done.

Section 1.4.2, entitled "The IDN Architecture", describes the types of IDN Processors and their relationships to the query processing functions.

1.4.1 QUERY PROCESSING

The query processing scenario presented below is initiated when the user enters a query into the system via a terminal attached to the IDN. In specifying the query, it is assumed that the user has no explicit knowledge of either the locations of the data requested or the data organization.

The query is sufficiently complex to require searching for two or more data values which may or may not share the same physical location. The data names used are referenced as local user terms. Qualifications are given to the data values to be considered for each of these names. One or more relationships are expressed between the data names.

The IDN does not specify the exact user interface. However, it assumes that the user interface does the minimal parsing to present the query in a "tokenized form." Although the tokenized form may be produced from a wide variety of user interfaces, it is most naturally associated with a forms-based query interface.

In the tokenized form, the query is organized into a hierarchy of "cells", which are rectangular areas within a form. Within a cell, data may be presented either horizontally or vertically. Cells may be nested within other cells.

Selection criteria are specified as boolean combinations of data qualifications associated with elements within each cell. The absence of any qualification is interpreted as an implicit "ALL" criterion (subject to security constraints). The hierarchical ordering of the cells enables the IDN to infer the record-types from which the elements must be selected. The IDN query decomposition process also infers how records must be joined in order to service the query.

Data qualifications include:

- Substring or pattern matching.
- Uniqueness tests.
- Relational operators (<, =, >, <=, >=).
- Boolean operators (AND, OR, XOR, NOT).
- Arithmetic expressions (+,-,*,/,()).
- Aggregate functions (COUNT, MIN, MAX, SUM, AVG).
- Set operations ("IN", <>).

The IDN's facilities to process user queries are divided into fourteen subsystems. These subsystems approximate the twelve steps for normal query processing outlined in Section 1.3.8. Two additional subsystems, one for supporting query development and one for monitoring and control of query processing, are also specified. Table 1.3 presents a list of these subsystems and the general metadata requirements implied by each. The subsystems are discussed separately in the following sections.

1.4.1.1 Subsystem 1. PROVIDE-LOCAL-DD-SUPPORT

An Information Resource Dictionary System (IRDS) provides local dictionary support to users at the User Node. This support includes:

- Help facilities.
- An inventory listing of available queries.
- The ability to create, save and recall stored queries.
- Definitions of:
  - Data element names (the Local User Name Spaces).
  - Derivation/computation names (e.g., TOTAL, SUM, MEAN).
  - System defined names, such as today's date.
- Information concerning default display format of output.

In addition to the above, the local IRDS provides a set of functions which allow for verification, security, and interfacing to the IDN query processing system.

The user receives support for examining local dictionary definitions, for composing queries, and for using stored queries. These facilities

QUERY PROCESSING FUNCTION	GENERAL METADATA REQUIREMENTS
1. Provide Local Dictionary Support.	Local-User-Name Space Network-Access-Name Space Stored Queries
2. Validate Tokenized Query.	Local-User-Name Space Network-Access-Name Space
3. Validate Intermediate Query.	Coherent Database Representations Query Control Block
4. Determine Data Accessibility.	Static Network View Dynamic Network View Query Control Block Coherent Database Representations
5. Decompose into Canonical Subqueries.	Coherent Database Representations Static Network View Dynamic Network View Query Control Block Subquery Control Blocks
6. Implement Execution Strategy	Static Network View Dynamic Network View Query Control Block Subquery Control Blocks Query Execution Graph
7. Query Monitoring and Control.	Static Network View Dynamic Network View Query Execution Graph

Table 1.3 - METADATA REQUIREMENTS FOR IDN QUERY PROCESSING FUNCTIONS - Page 1 of 2

QUERY PROCESSING FUNCTIONS	GENERAL METADATA REQUIREMENTS
8. Receive Subqueries at Destination.	Query Control Blocks Subquery Control Blocks Query Execution Graphs
9. Translate to Network Data Request Language (NDRL).	Subquery Control Blocks Coherent Database Representation Data-Base-Access-Name Space Database Subschemas
10. Extract and Compute Data (Execute NDRL).	Database Subschemas
11. Prepare Output for Network.	Subquery Control Blocks Query Control Blocks Query Execution Graph
12. Implement Composition Strategy.	Static Network View Dynamic Network View Query Execution Graphs Subresponse Header
13. Assemble Composite Response.	Query Control Blocks Subquery Control Blocks Query Execution Graphs
14. Deliver Composite Response and Present to the User.	Query Control Blocks Query Execution Graphs

Table 1.3 - METADATA REQUIREMENTS FOR IDN QUERY PROCESSING FUNCTIONS - Page 2 of 2

assure that, at any User Node, a user will be able to become familiar with the vocabulary to be used in queries and the procedures for formulating successful queries.

The User Node has access to the Local User Name Space (LUNS) and other information stored in the dictionary. For security reasons, specific detailed data element location and domain information is not maintained at the User Node or made available to it.

1.4.1.2 Subsystem 2. VALIDATE-TOKENIZED-QUERY

This subsystem performs first order validation of the query. For this level of validation, no knowledge of the network or its databases is required. This subsystem simply determines if all of the names and operations expressed in the query are known to the system and that they are used properly. In particular, it performs the following operations on the query:

- Validate all local names.
- Translate local names into system names.
- Identify "computational" names (such as SUM, TOTAL) and associate the required computations (if locally defined).
- Identify "standard" names (such as DATE, TIME) and distinguish them from local names.

The query which is input to this subsystem may be either a stored query or an ad hoc query. Stored queries may have previously received this level of validation. If a stored query has been prevalidated and not subsequently modified, this subsystem forwards the query to Subsystem 3, VALIDATE-INTERMEDIATE-QUERY.

If the query submitted is an ad hoc query or a stored query which has either not been prevalidated or has been subsequently modified, the query is in "tokenized form". The tokenized query is the result of passing lexical analysis. In this case, a symbol table is built; the query syntax is modified so that symbol table references replace the user-supplied labels, and the local dictionary supplies the translation between all local and system names. The result of this transformation is called the "intermediate form" of the query.

The names in the symbol table must be categorized: data element name, computation name, or standard name. (Internally, both computational names and standard names are FUNCTIONs, and are defined as such in the local IRDS. The only difference between the two is that the "standard name functions", such as DATE and TIME, require no arguments.) Names which do not fall into one of these categories are presumed to be local to the query. Once the names in the symbol table have been categorized, the correctness of their usage is ascertained.

The local dictionary supplies the external representation of data elements and the default format attributes for all required output values when not explicitly supplied by the user. Unresolvable conflicts in data types usage and sizing are identified.

If the query passes this level of validation, the intermediate form of the query is forwarded to Subsystem 3, VALIDATE-INTERMEDIATE-QUERY. Otherwise, error messages are returned to the user and the query is terminated.

1.4.1.3 Subsystem 3. VALIDATE-INTERMEDIATE-QUERY

This subsystem begins the process of Second Order Validation of the query. Using all of the network metadata and Coherent Database Representations (CDRs) makes complete validation of the query possible. Validation can be completed with respect to names, data qualifications,

and data relationships. The reasonableness of the query is determined based on the cellular structure of the query and the functional dependencies between data elements. An attempt is made to discover all possible errors before reporting back to the user. The following types of tests may be made:

- Value constraints (particular values or ranges).
- Location constraints (i.e., one or more names used in the query must be available from a particular "location", which may mean a subnet, a node, or an organization).
- Data grouping constraints (i.e., two or more names used in the query must be available in the same database, file, or record; this is a "context constraint").

This subsystem also begins the construction of the QCB by building the QCB header and message area for the query. This initial QCB is passed on to Subsystems 4 and 5 for completion.

1.4.1.4 Subsystem 4. DETERMINE-DATA-ACCESSIBILITY

This subsystem determines the accessibility of the data requested in the query. The elements specified in the query are used to determine the record-types to be accessed, and subsequently, the CDRs which correspond to the databases to be accessed. Security constraints are examined and any CDRs, record-types, or elements which are inaccessible because of security constraints are filtered out. Any potential security violations are identified. If no such violations have been identified, then the target database corresponding to each CDR referenced in the query is identified.

1.4.1.5 Subsystem 5. DECOMPOSE-INTO-CANONICAL-SUBQUERIES

Once accessibility has been determined, the decomposition of the query into subqueries must be performed. Subquery Control Blocks (SQCBs) are generated. Each SQCB consists of header information, a block of retrieval instructions, and input and output file descriptions. In general, a subquery produces an output file corresponding to a single relation. However, a subquery may have multiple input relations. Subqueries may process database relations exclusively, relations in files output by other subqueries, or a combination of these. The SQCBs are chained together via file descriptions to form the Query Execution Graph (QEG). (In the QEG, SQCBs are nodes. An arc is a file description which defines the file output by one subquery which is input to another subquery. All paths in the QEG eventually lead to a single node corresponding to the final response preparation.)

If there is no error in the generation of the QEG, the QCB is saved at the node where the generation occurred. This node becomes the "controlling node" for the query. The QCB is then shipped to "shadow" nodes, one of which will assume control if the controlling node fails during the execution of the query.

1.4.1.6 Subsystem 6. IMPLEMENT-EXECUTION-STRATEGY

The controlling IDN Node dispatches the subqueries to their target Data Nodes and awaits responses. It is responsible for the execution strategy of the query and for composition of the response. These functions are facilitated through the use of a Query Execution (and composition) Graph (QEG) produced at query decomposition time. This graph acts much like a "PERT" scheduling network for controlling the distribution of subqueries and the consolidation of responses. A "critical path" is determined for the processing of the individual query. This serves as a control reference. The QEG is part of the QCB.

It is maintained at the IDN Nodes which process the query or its response(s) until either the consolidated response has been delivered to the user or the query terminates abnormally.

The development of an execution strategy requires a knowledge of the network based on both static and dynamic information. The static nature of the network can be maintained in the dictionary. The dynamic status information, however, must be derived continuously. This information may be available through the underlying network communication protocols. If it is not, then IDN software must be responsible for generating and maintaining this status information as described in Section 1.3.8.

In addition to maintaining the Query Control Block for the original query, each subquery has a "Subquery Control Block" which contains the composition instructions and which identifies a named node where partial compositions take place.

Intermediate nodes must have a sufficient storage quota to accept and pass the data through for subsequent composition. The target (User Node) is defined as the final subcomposition node, and therefore the final composition node. A quota for the response is applied to the final composition node.

1.4.1.7 Subsystem 7. QUERY-MONITORING-AND-CONTROL

This subsystem is responsible for coordinating all phases of query processing; monitoring the progress of the query, and responding to error conditions which might arise. The function of query monitoring and control requires:

- Communicating status information to dispatched subqueries and to users.

- Checking the volume of the response and comparing it to the storage quotas of the User Node and Processing Node.

A mechanism of distributed control rather than centralized control is utilized here. There are no "control" nodes which are responsible for the entire system. Rather, "control" nodes exist only with respect to individual queries. As subqueries execute, status messages are sent to the node controlling the query as a whole. The QEG is updated with the status information, both at the controlling and shadow nodes. However, messages which initiate, hold, restart and terminate subqueries only emanate from the controlling node for the corresponding query.

The QUERY-MONITORING-AND-CONTROL Subsystem determines what action to take in the event of various exception conditions. These exception conditions include:

- A subquery exceeding output quota.
- The failure of a subquery.
- A timeout condition.

If a timeout occurs for any particular subquery or the query as a whole, then it may be desirable to stop the processing of the query and restart it from the point where the subqueries were transmitted. An option may be offered to the user to either terminate the query, restart it, or be sent a partial or "incomplete" response. If the processing was terminated due to security reasons, the user will not be given the option to receive a partial response.

Finally, this subsystem analyses the degree of traffic in the network. Should traffic exceed predefined levels, this subsystem will purge lower precedence messages from queues and place lower precedence subqueries on hold.

1.4.1.8 Subsystem 8. RECEIVE-SUBQUERIES-AT-DESTINATION

The "destination" for all subqueries is an IDN Node which is linked to a particular Data Node. This IDN Node serves as the interface between the Data Node and the network just as other IDN Nodes serve to interface the User Nodes to the networks. Since the node must be familiar with the particular "personality" of its Data Node, it is probably best that a particular destination IDN Node serve only a single Data Node. A backup node might be designated, however, in addition to the primary interface node.

This is a relatively minor step, but requires a great deal of support from the network communication subsystem. The subquery is dispatched to the host interface node where the subquery is to be processed for submission to the database at that host. The principal concern at this level of the system is failure protection through managed redundancy.

1.4.1.9 Subsystem 9. TRANSLATE-TO-NETWORK-DATA-REQUEST-LANGUAGE (NDRL)

This subsystem reformulates the subquery to access target database record types and elements. The reformulated subquery is in a source form called Network Data Request Language (NDRL). This language is a generic query language, with capabilities common to all database query languages. This transformation eases the eventual translation of the subquery to the query language of the target database's DBMS.

Each subquery was generated assuming that third normal form relations are to be accessed. In practice, however, target databases may be neither normalized nor relational. Thus the record-types associated with the target database may not correspond exactly to the record-types identified in the CDRs. Semantic errors encountered at this point would

indicate that the correspondence between the target database's schema and its CDR has not been properly defined.

1.4.1.10 Subsystem 10. EXTRACT-AND-COMPUTE (Execute the NDRL)

The subquery in NDRL format is passed to the Data Node (host processor). Here the NDRL is transformed into the syntax of a local query language or report writer. A separate execution module would be required for each different report writer language, but each module would accept the NDRL as input.

The processing of the subqueries received at the Data Node against its databases is similar to the manner in which a local query would be processed. The subqueries are already in the form of the standard query language, thus they can be submitted directly to the local execution facility for the given DBMS. The response received will be in a format controlled by the local DBMS. This response will be processed at the Data Node or sent to the designated IDN Node for translation into canonical form and integration with other responses.

1.4.1.11 Subsystem 11. PREPARE-OUTPUT-FOR-NETWORK

This Subsystem operates on the output from the local database's query language into IDN standard output form. In this form, extra space is removed from individual fields. Standard field-separator and record-separator characters are inserted, and appropriate headers are attached.

1.4.1.12 Subsystem 12. IMPLEMENT-COMPOSITION-STRATEGY

The output of any subquery will fall into one of three categories:

- It is part of the final response.

- It is input to another subquery at the same Data Node.
- It is input to another subquery at another Data Node.

This subsystem distinguishes among these cases. In those cases where a subquery processes output from subqueries from several nodes, this subsystem decides which node is the appropriate composition node. Most of the time this composition involves joining output relations. In these cases, this subsystem decides which join strategy is the most advantageous. It then directs output to the appropriate nodes and initiates subsequent subqueries.

1.4.1.13 Subsystem 13. ASSEMBLE-COMPOSITE-RESPONSE

This subsystem accumulates, sorts and merges all subresponses to form a single composite response to be sent to the User Node. It notifies the controlling IDN Node that the composite response is ready for delivery.

1.4.1.14 Subsystem 14. DELIVER-COMPOSITE-RESPONSE

This subsystem completes the processing of the query by:

- Inserting in the composite response any mapping information required by the User Node to present the response to the user.
- Performing any final computations.
- Delivering the composite response to the User Node.
- Informing the controlling IDN node that the response has been completed.

1.4.2 THE IDN ARCHITECTURE

This section presents the IDN Architecture which has been designed to perform all of the query processing steps presented in Section 1.4.1 in a manner consistent with the stated goals of the IDN. This architecture is the result of a careful and thorough analysis of the functional requirements and an evaluation of numerous alternative designs. Given the initial set of assumptions, the design has been determined by an analysis of where the basic query processing functions should take place and what metadata and data must be used to perform them.

A family of special purpose Dictionary/Directory/Query (D/D/Q) Processors has evolved which perform the required functions and maintain the necessary metadata. If required, responsibility for network monitoring can be assigned to some of these processors. The relationship between these "IDN Processors" and the query processing functions will be described in this section. How these functions are to be implemented will not be considered at this time except from the point of view of general feasibility, alternatives for implementation, and tradeoffs. It is also important to ascertain and estimate the impact of this approach across a network of existing User and Data Nodes.

1.4.2.1 The IDN Processors

From analysis of the IDN goals, requirements, and query processing steps to be supported by the IDN, an "Integrated Family of IDN Processors" has been synthesized. The User Nodes and Data Nodes alone are simply not sufficient to support the requirements of the IDN. Special IDN Nodes must be introduced to provide control and homogeneity in an otherwise heterogeneous network. Instead of having to develop custom query processing software for a variety of different processors, this approach should use uniform Information Resource Dictionary System and Query Processing software to run on a set of similar processors of the same general architecture but differing configurations. The maintenance of

the software and the reliability and longevity of the IDN are thereby greatly enhanced. This approach ensures not only that the IDN software is well "integrated", but also that the existing network of User and Data Nodes becomes "integrated" as well.

The complementary set of IDN Processors may be viewed as a hierarchy of three levels of query processing. At the "Interface Level", a D/D/Q Processor of type **D1** accepts the query from the user, performs first order validation and presents the response. A similar processor of type **D1*** receives subqueries directed to Data Nodes, accepts the response from the Data Node, and composes responses. At the "Network Level", the D/D/Q Processor of type **D2Q** completes query validation, dispatches subqueries, and controls query execution. Both the interface and network level functions may be assumed by an IDN processor at the "Backup Level". Here, the D/D/Q Processor of type **D3Q** can perform all of the functions of the D1, D1* and D2Q Processors as well as provide backup services for the Data Nodes. This arrangement provides for the greatest flexibility and permits a given machine to assume multiple roles as needed. Depending upon its capacity, a D/D/Q machine may provide user terminal "Interface Level" support, intermediate "Network Level" support, or complete "Backup Level" capabilities. An example of how these processors might be configured in a network is given in Figure 1.1.

The D/D/Q Processors proposed for the IDN Nodes should not be viewed simply as general purpose computers which run the Dictionary/Directory/Query software required to support IDN Query Processing. A more useful perspective would be to consider the D/D/Q Processors as special-purpose machines with hardware, firmware, and software, all designed to efficiently handle queries and responses and the required supporting metadata management in a manner consistent with the required secure environment. Like database machines, these "Dictionary Machines" would be optimized for their specialized tasks. Unlike most database machines, the D/D/Q Processors would be managing a base of relatively non-volatile information (the data and network metadata) and thus could take full advantage of data pages maintained in primary or cache memory.

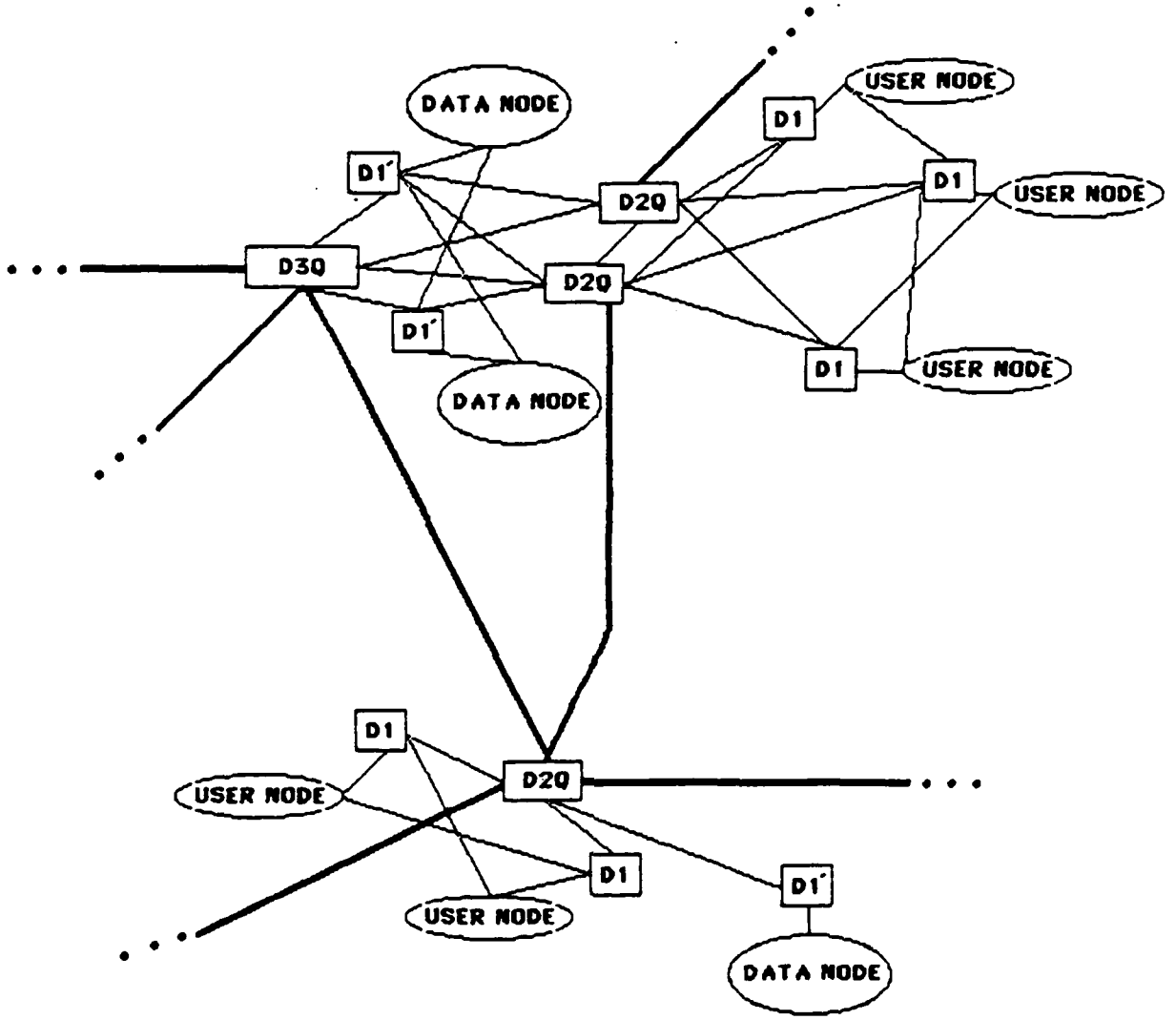


Figure 1.1 Example of a Section of the IDN

These processors would be a natural application for associative or content-addressable memories since data loading would be relatively infrequent. The required high-speed access to metadata might be difficult to achieve with general purpose computers, but could be obtained with special D/D/Q hardware.

It is necessary at this point in the discussion to present certain assumptions concerning the IDN architecture:

- It is assumed that the query is entered through a User Node and that the User Node passes the query, in IDN standard form, along to its companion D1 Node for disposition. The User Node is also responsible for later formatting the response to the query and displaying messages to the user.

- It is assumed that the D1 Node performs some first order validation of the query and then determines which D2Q Node should process the query based upon the Static Network View and status messages received from other IDN Nodes.

- The D2Q Node maintains network metadata (The Dynamic Network View) including the states of the other nodes in the network in order to perform routing and optimize query processing. This data is presumably supplied by the Network Protocol. The D2Q Node uses this status information to determine data availability, decompose the query into subqueries and dispatch them to the Data Nodes.

- The subqueries are received by D1* interface nodes which serve as companions to the Data Nodes. These processors complete the translation of subqueries from canonical form into the Network Data Request Language (NDRL) and receive the responses from the Data Nodes.

- The Data Node receives the subquery in the form of the NDRL, obtains the data from the databases, and sends the response

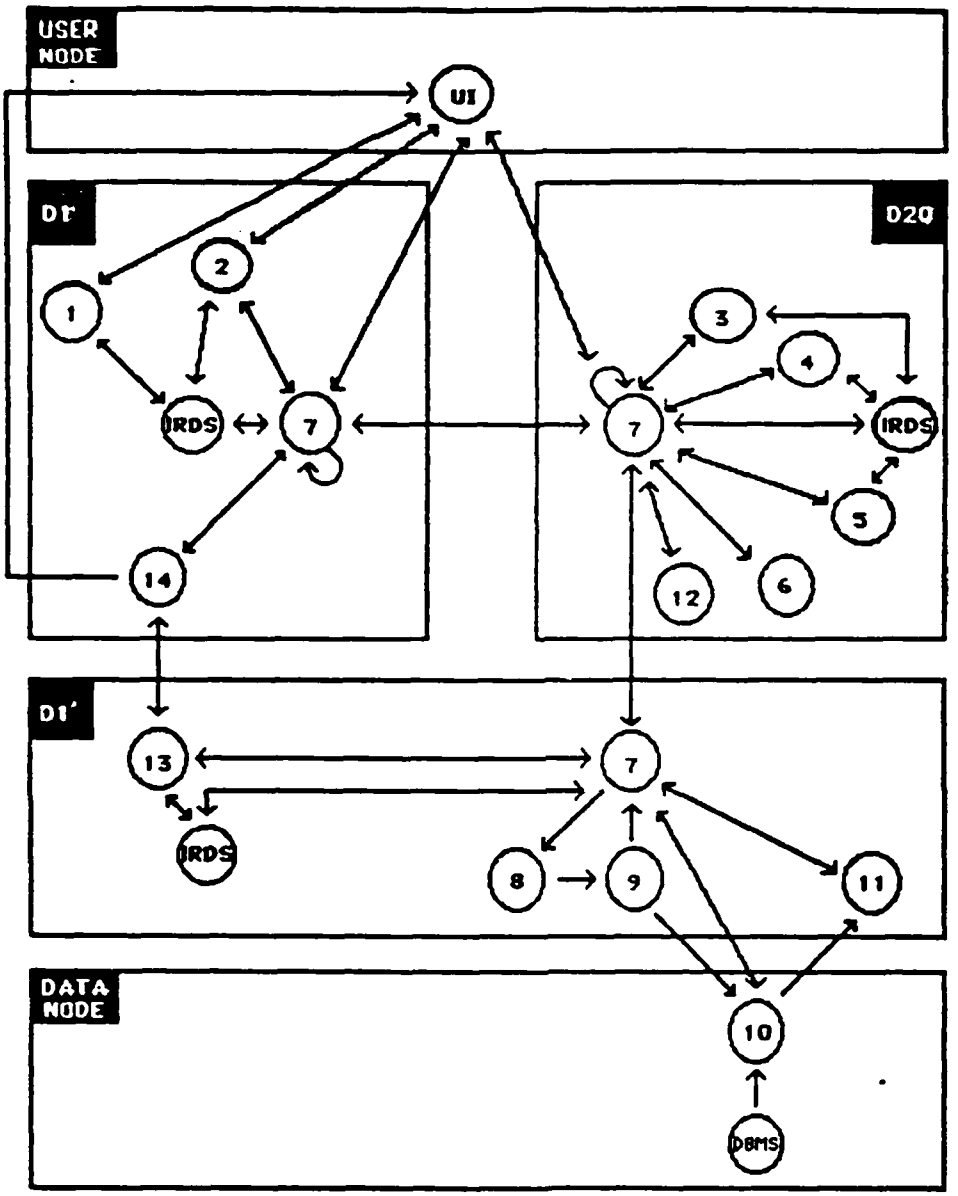
back to a designated D1* Node for composition. Any additional required translation into native query language may be performed at a Data Node, depending upon the target database(s).

- The rule is established that all public data which exists at a Data Node is replicated at its companion D3Q Node or some combination of other (D3Q and Data) Nodes. The D3Q Node thus serves as a backup for the Data Node if it is down or overloaded. Under normal conditions the D3Q Node provides the same functions as a D1 Node, a D2Q Node, and a D1* Node to the network users at the Data Node.

Figure 1.2 shows the placement of subsystems on IDN processors and the major communication paths between those subsystems. As can be seen in Figure 1.2, the D1 processor is dedicated to providing support for the development of user queries. The query validation performed at this node may be invoked by a user independent of attempting to execute the query. The D2Q processor has two major functions with respect to query processing. First, it performs the major part of query validation in decomposing the query into subqueries. Second, it controls execution f these subqueries. (Note that QUERY-MONITORING-AND-CONTROL actually has components on each processor. This assures that all decisions which can be made locally are handled locally.) Finally, it can be seen that the D1* processor is dedicated to controlling the execution of subqueries at the data node, and accumulating responses.

1.4.2.2 User Node Processing

The User Node receives the query from the user by means of a two dimensional input form and transforms it into a tokenized query that is acceptable to the IDN Nodes. The User Node is responsible for the final formatting of the response into a standard output form or a custom format specified by the user.



1 = Provide Local DD Support	9 = Translate to NDRL
2 = Validate Tokenized Query	10 = Extract and Compute Data
3 = Validate Normalized Query	11 = Prepare Output for Network
4 = Determine Data Accessibility	12 = Implement Composition Strategy
5 = Decompose into Subqueries	13 = Assemble Composite Response
6 = Implement Execution Strategy	14 = Deliver Composite Response
7 = Query Monitoring & Control	
8 = Receive Subqueries	

Figure 1.2

The User Node also provides the facility to capture the User's interaction with the dictionary at the supporting D1 node(s) and display the results of dictionary queries and commands.

1.4.2.3 D1 Node Processing

The D1 Nodes serve two primary functions:

- They relieve the User Nodes from all responsibility for managing user queries and supporting user development of new queries.
- They perform first-order validation of queries.

The User Node has access to the LUNS and other information stored in the D1 dictionary. For security and optimization reasons, it is not desirable to maintain specific detailed data element location and domain information at the User Node or the D1 Node. The D2Q and D3Q Nodes take responsibility for all location information and other sensitive metadata.

In general, a User Node will have a primary D1 Node to provide this support. In order to enhance robustness, D1 nodes may be backed up by other D1 Nodes. The dictionary in support of a given User Node at a backup D1 Node is kept synchronized with the primary D1 Node for the User Node. Note that the backup D1 Node for a given User Node will be the primary D1 Node for another User Node. Also note that a given D1 Node may support multiple User Nodes.

1.4.2.4 D2Q Node Processing

The D2Q Node is a special processor designed to provide the following functions:

- Second order validation of the query. In particular, this validation includes determining if the query "makes sense", and checking for security violations.

- Decomposing each query into canonical subqueries.

- Dispatching subqueries.

- Monitoring subquery execution and responses.

- Transmitting instructions to D1* nodes to initiate, terminate, hold, and restart queries.

- Determining response destinations and how to optimally join responses.

- Maintaining network status information.

Normally, D2Q Nodes will not be directly involved in composing responses. However, given appropriate conditions, they may take on that function.

The D2Q Node also provides the IRDS functions required to support these actions. The D2Q Node, therefore, relieves the User Nodes and Data Nodes of most of the added burden of network query processing and provides a common, homogeneous solution to the decentralized database problem.

D2Q Nodes are all "peers" to the network. However, for a given query a D2Q will become the "controlling node". The controlling node for a given query is the one which performed the query decomposition, which is where the QCB is generated. Once QCB generation is completed, the QCB is shipped to other D2Q Nodes. These become the "shadow" D2Q nodes. The QCB for the query will be maintained at all D2Q Nodes, but control messages for subqueries shall emanate only from the the controlling D2Q.

Should the control D2Q fail, one of the shadow D2Q Nodes will take over the role. The QCB identifies the order in which shadow D2Qs will be selected to assume control.

1.4.2.5 D1* Node Processing

A D1* interface processor serves as the destination node for all subqueries directed to a particular Data Node and receives the responses from this Data Node. The D1* Node performs the final translation of subqueries in canonical form into the Network Data Request Language (NDRL) and is responsible for any additional formatting of the responses which might be required in order to put them into network standard form.

In general, the D1* Node will be responsible for:

- Maintaining details of the database schemas and their data base management systems in order to support subquery translation.
- Performing some formatting of the responses from subqueries.
- Performing some calculations on the data (such as summarization).
- Executing the SELECT, PROJECT, and JOIN operations on the raw data derived from the databases, depending upon the capabilities of the Data Node it is supporting.
- Accumulating subresponses and preparing the final response.

1.4.2.6 D3Q Node Processing

These versatile nodes serve as companions to the Data Nodes. They have multiple roles:

- As backups for the Data Nodes.
- As repositories for replicated databases which can serve as alternate sources.
- As D1 Nodes for local queries directed to the network.

The importance of the D3Q Nodes may dictate that responsibility for their maintenance lay with the IDN Data and Network Administrators. Requiring local agencies to back up their own databases periodically may be inadequate. Backups may also be distributed across more than one D3Q Node, especially if the data can be naturally partitioned. Finally, there is a need to synchronize replicated databases with the original databases using some method such as date-time stamps. The mechanisms required for implementing a synchronization procedure need to be established for the Data Nodes and the D3Q Nodes.

1.4.2.7 Data Node Processing

The processing of the subqueries received at the Data Node against its databases is identical to the manner in which a local query would be processed. The subqueries have already been translated to the NDRL by the companion D1* interface processor for the target Data Node, thus they can be submitted directly to the local DBMS. Unless further translation into local query language must be made, this may be performed with the aid of a custom interface to the query facility of the local DBMS. In either case, the response received will be in a format controlled by the local DBMS. This response will be sent to the designated D1* Node for translation into network standard form and integration with other responses.

The composition of responses to subqueries into a single report may require the controlling IDN Node to send the results of one subquery to another Data Node or IDN Node so that it may be compared to the results

of another subquery, or even used in formulating a subsequent subquery. If this the case, it is necessary to stagger the processing of subqueries in order to ensure that processing at the Data Nodes was performed in the proper order. This is controlled by the Query Execution Graph established for subquery control and response composition.

(This page left blank intentionally)

1.5 SUMMARY AND CONCLUSIONS

The discussion of query processing and the proposed architecture of the Integrated Data Network has demonstrated the versatility of the design. Although some compromises were required, the goals of reliability, survivability, timeliness, low operational cost, and acceptable impact can all be satisfied. It is particularly important to recognize that these goals could be satisfied by allocating a small number of IDN Nodes to "clusters" of User Nodes and Data Nodes rather than replicating all data and metadata throughout the network. Not only is this approach consistent with the design goals, but it offers ease of implementation as well.

The concept of the IDN Processors as a family of three different levels of processors (i.e., the Interface, Network, and Backup levels), all having the same basic architecture, provides the flexibility and power needed to provide timely and reliable query processing. The impact of this solution upon the existing network is minimal and the operational cost of the existing nodes is only marginally increased. The survivability of the query processing capability is considerable, due to the built-in redundancy of the IDN Nodes and the ease of replacement of any given node. The multiple roles which can be played by the IDN Nodes makes it convenient to associate them with clusters of other nodes and reduces the number of special nodes required.

When the IDN Processors are viewed as specialized processors, rather than just general purpose computers executing query processing software, the concept presented here takes on an even greater significance. The performance which can be achieved with such specialization can be used to ensure that the network remains responsive even under widely varying loads and conditions. The static nature of the metadata permits it to be established in special types of memories (such as associative

memories) to achieve high-speed search. This efficiency could ultimately allow IDN query processing to go beyond simple database searches. Advanced versions of the system could employ intelligent search strategies based upon data classifications as well as data element names and relationships. The IDN Processors would permit the capabilities of the network to grow in order to meet the increasingly greater needs of selective information retrieval.

The IDN Node design described herein holds great promise as a potential solution to the query processing problem for a network of heterogeneous DBMSs.

CHAPTER 2

SUPPORTING TECHNICAL PAPERS

This chapter contains four technical papers that discuss issues that pertain to the specification of the IDN Subsystems which are presented in Chapter 3.

(This page intentionally left blank)

2.1 RESPONSE QUOTAS IN THE IDN

Analysis of the handling of subresponses and responses by the IDN has revealed that a more complete mechanism than was described in the Functional Description is needed for establishing output quotas for IDN users. Such a mechanism is necessary if control is to be maintained over the volume of output which may be generated by a single query.

Given the heterogeneous and decentralized nature of the target databases, it is difficult to predict the volume of a subresponse in advance. It is even more difficult to predict the volume of the final response since SELECT and JOIN operations against subresponses may greatly reduce the volume of the final output. Even so, the IDN must impose quotas on the final response and, perhaps, on the subresponses as well.

2.1.1 Multiple Thresholds of User Quotas

It is suggested that, at least, a "two threshold" quota system be established. These thresholds are:

- First threshold - a maximum level for individual subresponses.
- Second threshold - a maximum level for the final response.

It may be desirable to establish an additional threshold for a "warning", which could be set at some reasonable level, e.g. 100 lines of output. The warning would permit the user to determine if the output

being generated exceeds reasonable expectations. The second threshold should be established at a high level, e.g., 1000 lines of output, while the first may be established at some lower level, say 500 lines.

Since the volume of a given subresponse may greatly exceed the volume of the final response once response composition has completed, it is difficult to provide any meaningful warning regarding output volume from subresponses. Most of the overhead of query processing, however, involves the processing and communication of subresponses across nodes; therefore, some upper limit must be set on the resultant output of a subquery. A subresponse threshold of perhaps 10 times the final response limit may allow most queries to be processed while halting those which require excessive overhead. The subresponse quota could also be derived as a percentage of the current size of the target database, although it may be difficult to maintain estimates of database sizes.

2.1.2 Establishing User Quotas Based Upon Precedence

Each message will be tagged with the "precedence" or priority level which the user assigns to the submitted query. Rather than establishing quotas on a user-by-user basis, it may be advantageous to assign them based upon precedence level. Assuming, for example, that the IDN supports five precedence levels, the following table might be used to establish quotas (in units of lines of final response output):

PRECEDENCE	QUOTA
1. ECP / CRITIC	2000
2. FLASH	1500
3. IMMEDIATE	1000
4. PRIORITY	500
5. ROUTINE	250

Individual quotas may still be established on a user-by-user basis, particularly if it is desirable to make the quota for an individual different from that which would be associated with the user's precedence level. In some cases larger, rather than smaller, quotas might be assigned to allow for the types of queries and responses which are routine for the particular user.

The primary advantage of relating quotas to precedence levels can be seen when it becomes necessary for the QUERY-MONITORING-AND-CONTROL Subsystem to give priority to a particular query and its associated subqueries and subresponses. If resources are available, then the quota system simply helps to manage IDN overhead, but if resources are very limited (such as in a crisis situation) then precedence levels and their associated quotas can be used for increasing throughput for critical queries. Three scenarios will be considered:

- Special Processing for "Urgent" queries.
- Purging Queues Based upon Precedence.
- Dispatching Based upon Precedence.

2.1.2.1 Special Processing for Urgent Queries

It is essential that the IDN be able to process high precedence level queries through the system as quickly as possible. To accomplish this the IDN must manage its resources in order to permit expeditious handling of such queries. In addition to placing high priority queries at the top of input queues for such processes as query decomposition, it is also important the the IDN reserve sufficient space in its output queues for handling subresponses and responses.

In order to accomplish this, the QUERY-MONITORING-AND-CONTROL Subsystem must signal each of the D1* Nodes for each of the target Data Nodes to reserve space based upon the quota established for the high precedence query. This should occur as soon as a Query Monitoring and Control subtask has been established for the Subquery. The QUERY-MONITORING-AND-CONTROL Subsystem should also signal the "assembly node" so that it may reserve sufficient space to handle the composition of the response. The determination of the assembly point is dynamic, hence, this signal will have to be delayed until some subresponses have been generated.

2.1.2.2 Purging Queues Based on Precedence

If there is insufficient space in an output or processing queue to contain a subresponse or response for a high precedence level query, space will be provided by purging the queues of lower priority subresponses to make room for the response to higher priority request(s). When allocating space, if an individual user's quota is larger than that associated with the priority level, the higher quota will be used. Queries whose output has been purged will be put on "hold" until resources are available to process them. If all subresponses for a particular subquery must be purged, then the QUERY-MONITORING-AND-CONTROL Subsystem will notify the user that the query must be restarted.

Except perhaps for the highest priority queries, the decision to purge a subresponse should not rely on precedence alone, since this approach could result in some queries never being processed. The Query Execution Graph may be used to determine the impact of purging a particular subresponse from a given query. For a given subresponse, the total number of serial processing steps which preceded the subresponse can be calculated. This number can be used to discriminate among queries of equal precedence to determine which ones should be purged based upon the processing which has already occurred. This number might also be taken into account when deciding whether or not to purge responses for the next higher precedence level, i.e., a subquery which has undergone some number of processing steps may have the effect of being raised one level in precedence.

This strategy helps to insure the overall efficiency of IDN query processing, but it does not insure that all queries will ultimately be processed. Some time limit must be placed upon how long a query can be "held" within the system waiting for its processing to be completed. After some established time period, the priority of a given query could be raised by one level. Thus, after one or more time periods have passed, the query would eventually have a high enough priority to be processed. Likewise, this would limit the number of times that a query would be purged and restarted by the system.

2.1.2.3 Dispatching Based on Precedence

If a given output queue contains subresponses from several different queries, a mechanism must be established for dispatching these subresponses to the next Subsystem based upon query precedence level. The mechanism can be similar to that used in Section 2.1.2.2, in that both the precedence of the query and the number of processing steps used to produce the output can be used to determine the order of dispatching from the queue.

Again, it is necessary to include an "accumulated time" mechanism to insure that all subresponses are eventually dispatched even if they are of low priority. It is also necessary to insure that the ASSEMBLE-COMPOSITE-RESPONSE Subsystem does not have to "wait" too long for a given subresponse since this would cause a processing bottleneck. The QUERY-MONITORING-AND-CONTROL Subsystem should increase the priority of queued subresponses in an incremental manner as other subresponses are received by the ASSEMBLE-COMPOSITE-RESPONSE Subsystem for processing. This approach insures that the assembly process will not have to maintain multiple subresponses indefinitely while waiting on the single subresponse which is necessary to complete composition.

2.2 QUERY PROCESSING RESTART AND RECOVERY

2.2.1 Introduction

The requirements for the restart and recovery of queries within the IDN can be addressed in a straightforward manner by taking into account the costs incurred in the processing of a query at each stage and the level of redundancy of each component of the network. Although a broad spectrum of options are available, the practical considerations of the system design guide the analysis toward a solution which is consistent with the IDN as described in the System/Subsystems Specification.

In considering the recovery process within the network several important observations can be made. First, one must decide the level at which recovery must take place. It is possible to operate the network at a level where there does not exist an inherent recovery mechanism; and the user would have the capability to ask the system the status of a current query. In this environment the user would submit a query, and then from time to time access the system to see if the query was still active. If the query was not progressing, the user could request a restart; this action would then effectively terminate the old query. The only requirement for this methodology would be to keep track of the status of a particular query in its processing schedule. The mechanism for garbage clean up would involve a time stamp attached to each fragment of a query. At some point, all fragments with a time stamp exceeding a certain value would be eliminated. This would prevent residue buildup within the network under this simplistic scheme.

At the other end of the recovery spectrum, the IDN environment could process every query under every condition of failure, except perhaps a massive failure above some threshold level of network resources. This implies a completely different approach to recovery contrasted to the

simplistic approach, as both the network elements and the transactions within the network must be failsafe. In order to make the network elements failsafe D1, D1*, and D2Q elements would be built around failsafe architectures, and such architectures would include multiple CPUs, memories, buses, power supplies, etc. Further, the IDN would necessarily contain multiple paths between any nodes to preclude isolation due to path failure. Under any single point failure of hardware and several cases of multiple hardware failure, the transaction is then guaranteed to be completely failsafe. Since hardware integrity of the IDN is guaranteed, the only difficulty remains the software quality within the IDN. This mechanism would not guarantee the availability of the database, since it may reside on a non-failsafe environment and it may exist on a unique machine.

Within the spectrum which has been briefly discussed at this point, a compromise position has been selected upon which a failure/recovery mechanism is to be built. It is assumed that the various hardware components, the D1, D1*, D2Q and all other systems are not failsafe hardware. The position selected for failure/recovery is essentially equivalent to a **point of no return.** That is, if it is less costly to resubmit the transaction, then it will be required that resubmission be the recovery process. If the point of no return is passed, then the IDN system itself will have jurisdiction for recovery, except in cases of extreme failure where recovery is made impossible. In selecting this no return point, it has been attempted, conceptually, to balance the expended resource allocated to the query at this point to the cost of performing the recovery. The optional point of no return appears to be the moment when subqueries are dispatched from a D2Q to begin the process of gathering data. Up to that point only two processors, one D1 and one D2Q, are involved. As processing continues past this point, a large number of machines become involved in a potentially very complex manner. Hence, resubmissions may be far more costly than the associated recovery process.

The balance of this paper will discuss the recovery procedures from the point of no return. The process of query submission and execution follows the outline described in the other IDN papers in this Chapter.

2.2.2 A Review Of Query Processing

For purposes of this paper it is assumed that the processing of queries occurs as described in the System/Subsystems Specification. This document describes fourteen subsystems which are necessary to process a query. They are as follows:

1. Provide Local IRDS Support - IRDS facilities are made available at the User Node to support the composition and execution of queries.

2. Validate Tokenized Query - The query is accepted in tokenized form from the User Node and process of transformation into canonical form is begun. Here all labels are equated to their corresponding names in the NANS, and naming usage conflicts are identified. The normal output of this step is the normalized query.

3. Validate Intermediate Query - The normalized query is validated for correctness and completeness of semantics.

4. Determine Data Accessibility - Check node and net network status to determine is the required data accessible.

5. Decompose Query into Canonical Subqueries - If necessary, the query is divided into subqueries, each with its own target database. Here the Query Execution Graph is generated.

6. Implement the Execution Strategy - Initiate subqueries and maintain the status information in the Query Execution Graph.

7. Query Monitoring and Control - Monitor the progress of query execution; determine the volume and completeness of responses, and respond to error conditions which might arise.

8. Receive Subqueries (at Destination) - Receive the subqueries for processing against the databases at the Data Nodes.

9. Translate to Network Data Request Language (NDRL) - Translate the subqueries into a standard form for processing against the databases.

10. Extract and Compute Data (Execute the NDRL) - The subqueries are executed against the target databases producing responses.

11. Prepare Output for Network - Translate the responses into a standard network format for output.

12. Implement Composition Strategy - Designate appropriate site for response composition and transmit data.

13. Assemble Composite Response - Assemble the responses from the subqueries into a single composite response.

14. Deliver Composite Response - Transmit the composite response to the User Node an make it available for presentation.

As described in Section 2.2.1 of this paper the recovery process is not applicable until Subsystem 5 completes its processing of a query. That is, at the completion of Subsystem 5, the point of no return has been reached. Once this processing has completed, then, from Subsystems 6 through 14, the recovery process exists which permits queries to complete.

These fourteen subsystems use five primary processor types plus one type of optional processor type. These are again described in the System/Subsystems Specification. The processor types are D1, D2Q, D1*,

UN, and DN. The optional type, the D3Q, is a multi-role processor type which can serve as a Data Node, a D2Q, D1*, or D1. The actual realization of the IDN is not significant in terms of the recovery concept except that the processor types must eventually include replication of processors so that recovery can occur. In no case is it permitted that the IDN consists of a single machine of any particular type. The UN and DN nodes will not be considered further in recovery, as they may very well be unique in terms of information supply, and are both potentially outside of IDN configuration control. The D3Q node can be used as a Data Node and can also function as a D2Q, D1* and D1. Its presence is not necessary for IDN operation, and so it will not be considered as an essential component of the recovery process. Its presence will however be beneficial as it can be useful in terms of DN replication.

From the System/Subsystems Specification the node classifications and appropriate processing steps allocated to those nodes are given in the following table:

Subsystem Number	Primary Node
1, 2	D1
3, 4, 5, 6	D2Q
7	D1, D2Q, D1*
8, 9	D1*
10	DN
11	D1*
12	D2Q
13	D1*
14	D1*

Query Transaction Step with the Associated Processor Type

Note that Subsystem 7 is distributed among all nodes, and components which reside at nodes according to their type are different.

Note also that assembling responses is performed at a D1* node. In this scenario, D1* nodes with subresponses determine, based on subresponse size and resource availability, which node can assemble the response at lowest cost. It is assumed that if distance is a factor in determining communication costs, the differential in cost will not be significant enough to modify the composition strategy.

2.2.3 Definition Of A Query Failure With Gross Recovery Strategy

Using the material presented in Section 2.2.2, it is now possible to define a failure and an associated recovery procedure. The meaning of the failure of a query is clarified by the following definitions:

Definition 1 A **processor chain** is the set of processors which are used both sequentially and in parallel for the resolution of a query.

Definition 2 A query is said to be in a **failed state** if any processor in the chain is in a failed state.

Definition 3 A query in a failed state is said to be **recoverable** if the information lost by the failed processor in the chain is available by either replacement by recomputation or replacement from a backup resource.

Definition 4 The **invocation life** of a query is that period of time which begins with query submission to the D1 node and ends when the originating User Node receives the result, or ends when the user terminates the query through a cancellation mechanism, or when the IDN system itself terminates the query.

Definition 5 A query in a failed state may be **permanently disabled** if the information lost by a failed processor is not replaceable from any source during the invocation life of that query.

Using these definitions it is now possible to construct a matrix of query conditions with the understanding that a general processor chain would appear as follows:

UN; D1; D2Q; {D1*}(i/1); {DN}(j/1); {D1*}(k/1); D2Q; D1; UN (1)

Here the braces indicate multiple processors of the given type executing in parallel, and the ratios (e.g., i/1) indicate the maximum and minimum number of such processors which could be involved. Note that the processor chain in (1) takes into account replication of processors by type only to service a query and ignores restart considerations.

A matrix of possible recovery strategies is shown in Table 2. The contents of the matrix indicate the nature of the restart and recovery process for each processor type. This is defined in the paragraphs which follow.

Query Recovery Strategy by Processor Type

IDN Subsystem	D1	D1*	D2Q	UN	DN
1	RESBMT	NA	NA	RESBMT	NA
2	RESBMT	NA	NA	NA	NA
3	NA	NA	RESBMT	NA	NA
4	NA	NA	RESBMT	NA	NA
5	NA	NA	RESBMT	NA	NA
6	NA	NA	RECVRBLE	NA	NA
7	NA	NA	RECVRBLE	NA	NA
8	NA	RECVRBLE	NA	NA	NA
9	NA	RECVRBLE	NA	NA	NA

IDN Subsystem	D1	D1*	D2Q	UN	DN
10	NA	RECVRBLE	NA	NA	PDISABLED
11	NA	RECVRBLE	RECVRBLE	NA	NA
12	NA	RECVRBLE	RECVRBLE	NA	NA
13	RECVRBLE	NA	NA	NA	NA
14	RECVRBLE	NA	NA	HOLD	NA

In this Table it can be seen that the only point where a query can become permanently disabled is when the DN processor is in a failed state during Step 10 of the query. Also, there is a condition which requires a "HOLD". In this condition the IDN system simply holds the information until a routing instruction is received, or until the system set default time for a HOLD is reached and the system normally cancels the query.

Generally, in this Table it is assumed that only a single occurrence of a processor of a given type in the processor chain has failed. If multiple processor types in the processor chain fail, then the Table would necessarily be modified. However, with enough replication, the Table could remain identical for any number of processor type failures. Further analysis of recovery under multiple failures must be performed in the context of a particular topology for the IDN. At this time, sufficient redundancy must be assumed to support recovery under multiple failures.

In this Table, the D3Q, since it may act as any one of the other processor types, is not explicitly shown.

A final comment must be made regarding the interconnection scheme of the network. In describing the possible actions resulting from this Table, it is a requirement that the UN directly reach several D1s; each D1 directly reach several D2Qs; each D2Q directly reach several D1*s; and each DN be directly reachable by a minimum of two D1*s. This insures

multiple path structures which prevents single point failures from being catastrophic due to isolation of nodes such as the UN and DN.

2.2.4 The Recovery Scheme

The first requirement of the recovery scheme is to modify the processor chain of expression (1) to the following chain denoted (2):

```
UN; [D1](2); [D2Q](2); [D1*](i/1); [DN](j/1);                    (2)
    [D1*](k/1); [D2Q](2); [D1](2); UN
```

The square brackets denote processor types in the processor chain which are replicated for recovery purposes. The values in parentheses indicate the level of replication. The modification allows the replication of the D1 and D2Q processor types to be exactly two machines. The alternative machine is never activated, with respect to a given query, except when the failure of the primary machine is determined by a processor in the immediately preceding position in the processor chain (all processors may be actively processing some query at any one point in time). Of course, any chain of machines (UN; D1; D2Q; D1*; DN) in the network may be involved in the processing of any given query, its subqueries, its subresponses, and the response.

There are several schemes which can be proposed for determining that a failure has occurrence at a node or processor of a given type. The two most common techniques are: *1) a notification mechanism, and 2) a* required acknowledgement with or without timeout. In the first case a node is required to send a message to some determined point with a given frequency. If the message is not received, then it is assumed that the transmitting node is not available and notification is given within the network. This technique will not be considered further as it suffers from several fundamental problems, the least of which is that the *notification point itself must be distributed, which is an exponentially* compounded problem.

It will be assumed that the technique to be used in all communications is a required acknowledgement from any transmitted request. This technique is combined with a static model of worst case performance to establish time out thresholds. It is also assumed that the underlying protocols are able to distinguish a busy processor from a failed processor. (It is possible that a given set of queries may saturate one or more processors, even though the network as a whole is well below saturation level.)

The underlying structure of a transaction within the IDN is then that every request or command dispatched from a processor to another processor will require an acknowledgement within an a priori established timeout. If the timeout occurs, then the processor in the processor chain is assumed to be in a default status and the message is automatically retransmitted to the backup node. In this way, the backup node is automatically enabled simply by the arrival of a transaction coming from a processor type that timed out under transmission to the primary node.

In order to permit transaction transmission to the backup processor and to provide a cleanup process after failure, each Query Control Block (QCB), Subquery Control block (SQCB), and transaction has an identifier associated with it. A date/time stamp is part of this identifier.

When a processor of a given type is reactivated it can erase all transactions which have exceeded a certain age, or, equivalently, the restored processor can interrogate the controlling D2Q to determine a query status. If the QCB or the query log shows a new primary processor chain instance or if the query is marked completed or terminated, then the restored node would again erase all applicable elements. Due to the time lag between an event and its reporting, there is a period of indeterminacy. If a failure occurs, and the failed processor is brought up again in some short interval in time, then it is possible to have transactions being sent to both the primary and secondary processor. In order to avoid this problem, every processor will have a limit assigned which will not allow the processor to respond or accept transactions for

some fixed time limit. This guarantees that the indeterminate state can never be entered.

In order to understand the nature of the failure/recovery system, a brief step by step discussion of the procedures necessary to effect recovery is provided at this point:

- As defined in Subsystem 2, VALIDATE-TOKENIZED-QUERY, since this subsystem completes processing before the point of no return, there is no need to send the QCB to an alternate D1. This process does not take up much time relative to the query's invocation life. Even relative only to the complete validation of the query and generation of the Query Execution Graph, the exposure to failure here is minimal. Furthermore, the only time the user would be presented with any difficulty in resubmitting the query, would be in those cases where the query was not saved. As stated in the description of Subsystem 1, queries are saved both at the primary and alternate D1 Nodes assigned to a user node. It is also assumed here that if the user has submitted an ad hoc query, it is either very simple, or a modification of an existing stored query. This problem disappears if the User Node saves the last query submitted.

- Subsystem 5, DECOMPOSE-QUERY-INTO-CANONICAL-SUBQUERIES, provides for the shipping of the QCBs to alternate D2Qs after the QEG is completed. This approach implements the previously discussed "point of no return."

  A basic issue exists as to whether the assignment of alternate D2Qs should be done based on static or dynamic criteria. Given that the dictionaries at all the D2Qs are identical (except for the QCBs which are saved in ACTIVE-QUERY entities), it is conceivable that the assignment of alternate D2Qs could be done dynamically. This, however, introduces significant complexity into the assignment process. Furthermore, this approach would generate additional traffic. (For example, if the assignment were to be handled by a bidding process, bid requests, bid responses, and assignment

messages would have to generated.) Therefore, the simpler approach of assigning alternate D2Qs based on static criteria has been chosen. A simple relationship between IDN-NODE entities in the IDN dictionaries is sufficient to designate which nodes act as backup for other nodes.

Another issue is related to the degree of replication. Obviously, the more shadow nodes are designated, the greater the degree of reliability. However, this requires that processing as well as data be replicated. (An alternate D2Q Node must not only save the QCB generated at the primary, but also maintains the same status information in the QEG; hence the term "shadow D2Q".) Thus, as the degree of replication is increased, more powerful D2Q processors must be used to accomodate the additional processing and auxilliary storage required.

There is the issue of whether the alternate nodes should be identified in the QCB. Since a static assignment of alternate nodes is used, there is no compelling reason to do so. If the degree of backup is large, it would be more convenient to simply look up the alternates for the primary node as needed. However, in the case, where the number of alternates is small, adding fields to identify them in the QCB does not introduce significant overhead. Since a relatively low degree of replication is anticipated (two or three), this approach seems reasonable. Accordingly, the designation of the shadow D2Qs will be added to the QCB in addition to the primary. In order to avoid confusion as to which D2Q is the shadow in the event of a node failure, the D2Qs will be simply listed in a linear "pecking order". The controlling D2Q is always the first node in the pecking order.

- Subsystem 6, IMPLEMENT-EXECUTION-STRATEGY, initiates all subqueries, including those which are dependent on other subqueries, and maintains status information in the QEG; therefore, it must run concurrently on both the controlling and shadow D2Q Nodes.

An issue to be addressed is whether this subsystem is initiated after, or in parallel with, the sending of the QCBs to the shadow D2Q Nodes. In the former case, a modest delay is incurred because no subqueries are sent out until all shadow D2Qs acknowledge receipt of the QCB. In the latter case, subqueries may be started with one or more shadow D2Qs in a failed state. The latter case may be alleviated by simply increasing the degree of replication. As a compromise, it is proposed that a threshold level of replication also be designated. This threshold level is the minimal number of shadow D2Qs which must be active before any subqueries are initiated. It is proposed that Subsystem 6 be initiated at the D2Q once the controlling D2Q receives acknowledgement of receipt of QCB by the threshold number of shadow D2Qs. For high reliability the threshold level should be one; i.e., there is at least one shadow active prior to initiating subqueries. This value, like the degree of replication, should be adjustable.

- Once a subquery reaches the D1* processor, the procedure for recovery relies upon the redundancies among the D1*s and the data sources (DNs and D3Qs), rather than replication in the sense of "shadow nodes". Each D1* receives the SQCB for a given subquery (Subsystem 8), translates it to NDRL form (Subsystem 9), attempts to transmit the NDRL subquery to the designated DN and expects to receive the subresponse from the DN. The following recovery conditions may arise:

  - o The D1* receiving the subquery does not acknowledge receipt within the timeout period. The D2Q must transmit the subquery to an alternative D1* for the same DN if one is available. Otherwise, the D2Q must route the subquery to a D1* supporting a DN or D3Q which can supply the same data.

  - o The D1* acknowledges receipts of the subquery, translates it to NDRL form, and transmits it to the DN but the DN does not

acknowledge receipt within the timeout period. If another D1* is available for the same DN, a "local" decision is made by the original D1* to submit the NDRL subquery through the alternate D1*.

- o The DN does not acknowledge receipt of the subquery from the original D1* and does not acknowledge receipt from the alternate D1* (or, no alternate is available). Under these conditions, a "network level" routing decision must be made by the D2Q/C. The D1* informs the D2Q/C that the DN (or its communication lines) is down. The D2Q/C then determines if an alternate data source exists and routes the original SQCB from it to the D1* supporting the alternate DN (or,to a D3Q). The original SQCB must be used rather than the NDRL form since the new data source may have a different structure than the original source. If no alternate source for the data is available then the subquery must be cancelled, but other subqueries for the same query may still be processed.

- o The final condition to be considered occurs when the DN acknowledges receipt of the subquery to the D1*, but cannot return the subresponse to the same D1* due to unavailability of the processor or line failure. Here, the DN will attempt to send the subresponse to an alternate D1* over a redundant communications channel. If a D1* receives a subresponse which it did not originally transmit it must notify the D2Q/C so that subresponse composition can be properly controlled. If the DN cannot transmit the subresponse to an alternate D1*, then a timeout will ultimately occur for the subquery at the D2Q/C and the subquery will be cancelled.

- Failure of a DN allows recovery only if the Delegated Production Policy (DPP) identifies an alternate source for the required data, in which case the SQCB is routed to the alternate DN, or, a companion D3Q is available for the DN. Otherwise, the subquery is terminated by Query Monitoring and Control at the D2Q/C when it is

notified by the D1* that the DN is unavailable and that no other DN (or D3Q) with the required data is available. The termination of a single subquery does not, however, necessarily mean that the query is terminated. Related subqueries will continue to be processed (unless they were dependent upon input from the terminated subquery) and a partial response will be presented to the User Node.

- Once a query transaction, that is, its collection of subqueries, are complete and the results are assembled together as specified by Subsystem 13, ASSEMBLE-COMPOSITE-RESPONSE, then the process of transaction shutdown can begin. This process will become part of Subsystem 13. It involves notifying the reserve machines, the D2Q and the D1, that they are no longer needed for this query. To accomplish this, Subsystem 13 transmits a destruct message for the QCB and monitoring information at the respective processors. No acknowledgement is returned, since it is not critical that the backup systems actually receive the message. The reason for this is that at regular points, each system will inventory itself to clean up any old, outstanding elements. This process will involve both an aging criterion as well as the ability to check status of a transaction by requesting information from the designated D1 processor.

2.2.5 Summary

The proposed failsafe mechanism for the IDN is a passive system based on replication of certain key data elements at certain key places within the processing chain of a query. The failsafe mechanism does not come into operation until a particular amount of processing has taken place relative to the query. This point is referred to as the point of no return. Up to this point recovery is simply the process of resubmission of the query either in its original form, or in the processed QCB form. When subqueries are ready for issuance into the network, then the process if recovery comes into play.

An important characteristic of this scheme is that no active monitoring of node availability need occur beyond that which is provided by the underlying network protocol. When a response is not received after a message is transmitted, then that same message is automatically transmitted to the alternative system. If such a failure occurs in transmitting from a D1* to a DN and transmission from an alternate D1* to the same DN cannot be accomplished, then the D2Q/C routes the original SQCB to an alternate data source (a DN or D3Q) if one is available. Only when the D2Q/C determines that no data source is available to satisfy the subquery is the subquery cancelled. Other subqueries for the same query may still be processed in order to partially satisfy the query.

There are in this scheme several areas left open for consideration or decision at the time of implementation. These areas are dependent upon the nature of the IDN network software, the physical interconnection topology, and the presence of a processor or processors of type D3Q.

2.3 SUBQUERY PROCESSING AGAINST THE BASIC DATA MODELS

The subqueries processed by the IDN must be translated from Network Data Request Language (NDRL) format to the native format of the DBMS which controls the target databases before data can be retrieved. Although it is possible that a variety of commercial and custom DBMSs may be used at the various Data Nodes, these systems will probably use one of the following "basic" data models:

- Hierarchical.
- Network.
- Relational.
- Inverted File.

Each of these data models provides a different method for the representation of entities (data objects) in the database and the relationships among them. The property types associated with the entity and relationship types probably have similar representations although the data types used for value storage may differ. This paper will consider the particular requirements for subquery translation and processing for each of these data models.

2.3.1 The Phases of Query Translation and the NDRL

It is important to understand that queries are translated into the native format of the target DBMS, whatever the data model, through

several phases and not in a single step. The forms which the query takes during the translation process are:

- o Tokenized Query.
- o Intermediate Query.
- o Canonical Form Subquery.
- o NDRL Subquery.
- o Native Format Subquery.

The forms and the associated translation process are:

1. **Tokenized Form**

   The tokenized form is the IDN standard form used to pass the query from the User Node to the D1 Node.

2. **Intermediate Form**

   The data element names are translated from the Local User Name Space (LUNS) to the Network Access Name Space (NANS) and the query is validated prior to producing the intermediate form. This is the form of the query which is passed along to the D2Q/C (Controlling D2Q) Node for decomposition into canonical form subqueries.

3. **Canonical Form**

   The IDN canonical form is used for the subqueries and is suitable for passing them to the D1* Nodes, but it is an IDN internal form and it is not, itself, a database query language.

4. **NDRL Form**

The NDRL form of the subqueries are generated at the D1* Node by transforming the canonical form subqueries received from the D2Q/C. The data element names are translated from the NANS to the Database Access Name Space (DANS). The subquery is in the form of a generalized query language which can be sent to the Data Nodes.

5. **DBMS Native Form**

The NDRL form of the subquery is modified at the Data Node to conform to the data model of the target database and the syntax requirements of its DBMS interface. The subquery is now in DBMS native form and can be executed against the database.

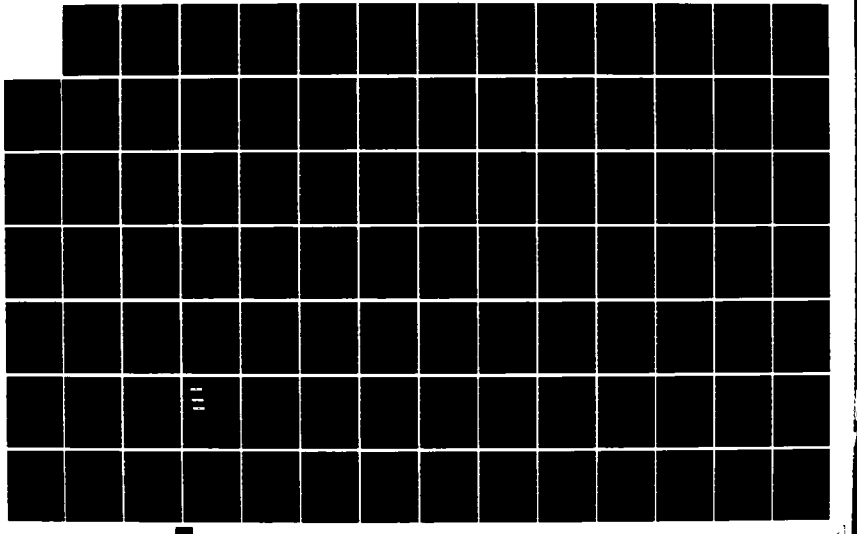
The first three forms of the query / subquery discussed above are IDN internal forms which, although they include the fundamental information needed for processing the query, are not suitable for execution by the Data Manipulation Language facility (DML), Data Query Facility, or Report Writer Facility of a given DBMS.

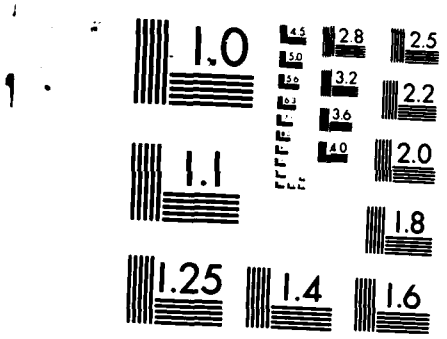
The fourth and fifth forms of the subquery are "external" forms and are intelligible both to humans and, in the case of the fifth form at least, to the DBMS. The database subschemas stored at the D1* Nodes are used to determine what additional clauses and names (e.g., record names) must be included in the subquery in order to process it against the target database. The NDRL is an external form of the subquery in that it is intelligible both to human operators and to the facilities of the target DBMS as well. For the NDRL subquery to process successfully, however, some additional syntax may be required.

Since the D1* is "aware" of the underlying data model for the target database, it can structure the NDRL subquery in a manner appropriate for that database. This structure will be somewhat different for each of

AD-A173 344 DESIGN CONCEPTS FOR DATA BASE UTILITIES(U) AGS SYSTEMS 2/3
CORP HARVARD MA APR 86 RADC-TR-86-48 F30602-84-C-0011

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

the basic data models. The NDRL could, therefore, be considered as four different languages if the different data model specific forms of subqueries are emphasized. It is better, however, to view the NDRL as a single language by emphasizing the elements which all subquery forms have in common. Such common elements include:

- SELECT clause - listing the data element access names (DANS) for which data values are requested.

- FROM clause(s) - indicating the names of the logical record(s) which contain the required data.

- WHERE clause - providing the selection or restriction criteria for the data values.

  - relational operators supported include: EQ, GT, LT, LTE, GTE, NE, EXISTS, FAILS.

  - substring comparisons and "wild card" operators (*, ?) are supported.

  - two or more criteria may be connected with logical operators (AND, OR, NOT).

- OUTPUT clause - specifying any special formatting requirements for the output; the output must be in tabular form with fixed column widths.

  - the ORDERED-BY subclause may be used to specify one or more data elements as sort keys; a CONTROL-BREAK subclause may also be used for sorting when summary calculations are to be performed.

- headings and footings must be turned off; no (non-blank) column separator characters should appear in the output; character-type data values should be left justified and numeric-type values should be right justified; numeric-type values should be in ASCII character (non-binary) format.

Even if the original query is rather complex, the NDRL will, in general, be rather simple since the subqueries have been reduced in complexity through the subquery decomposition process.

If all of the Data Nodes were to use the same Data Model (e.g., the relational model) and could be accessed by a generalized query language (e.g., an SQL-like language) then very little additional translation, if any, would be required to process the query. Since different data models and different DBMSs are used at the Data Nodes, some further translation is required. The subquery is transmitted in to NDRL format from the D1* to the Data Node and the translation to native DBMS format is completed at the Data Node itself.

Although the exact nature of the final syntax will depend upon the particular DBMS, the following general kinds of changes may occur during the transformation of the NDRL to native format:

- o SELECT may be changed to such verbs as PRINT, LIST, FIND, or LOCATE.

- o FROM may be changed to IN or ".", or, if the data element names are unique and associated with a single record-type, then record qualification is unnecessary.

- o WHERE may be used as is, or it may have to be combined with or replaced by an IF clause or similar restriction clause. The full range of conditions available with the NDRL may not be available for DBMS based on the hierarchical data model. If

hierarchical data model is the case then more general substitutions will have to be made in order to respond to the subquery (e.g., if a substring match cannot be accepted, then all values may have to be retrieved).

- o OUTPUT may have to be changed to such clauses as:
  - ORDER-BY for sorting on one or more data elements.
  - TITLE OFF to produce a null page and column headings.
  - PAGE OFF to eliminate page numbers and footings.

It is important to understand that the IDN assumes that a high-level ("English-like"), non-procedural, interface is available for each of the DBMSs at the Data Nodes. If the only interface available for a particular DBMS uses low-level primitives (such as FIND, GET, GET NEXT, GET NEXT WITHIN PARENT, HOLD), and if the interface is procedural in nature, then a special interface will have to be developed to translate the NDRL subqueries into the corresponding query procedures (generated application programs) at the Data Node.

Examples of the kinds of DBMS facilities which can support NDRL translation are:

- o Hierarchical DBMS - SYSTEM 2000: PLEX, QUEST, and QUE.
- o Network DBMS - IDMS: OnLine English, OnLine Query, and Culprit.
- o Relational DBMS - SQL/DS: ISQL, DBS, and CICS application programs.
- o Inverted File DBMS - Model 204: User Language and Host Language Interface.

The facilities mentioned above are intended as examples only. Facilities of other DBMSs may support NDRL equally well.

Examples of the kinds of DBMS facilities which would require interfaces capable of generating procedural programs in order to translate NDRL include:

- o IBM IMS and DL/1 Host Language Retrieval Commands.
- o Total Data Manipulation Language (DML).

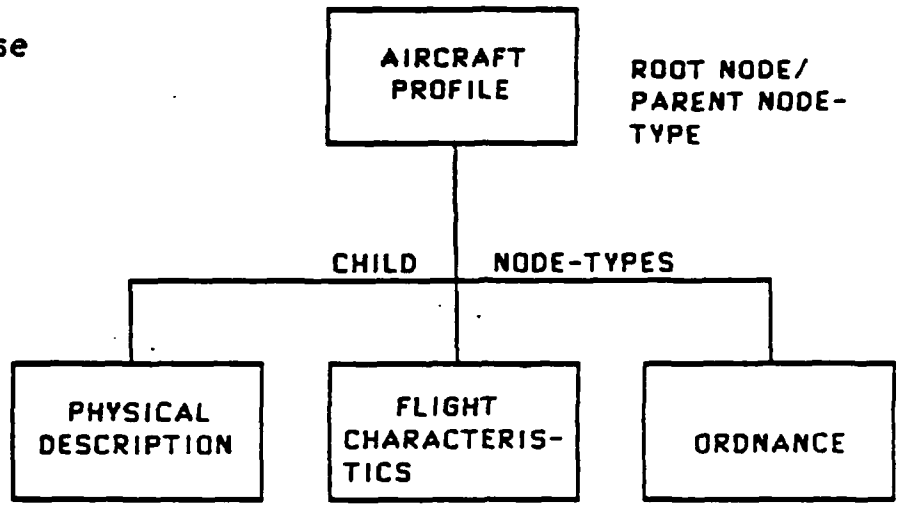
2.3.2 Subquery Processing Against the Hierarchical Data Model

The hierarchical data model is in many respects a simple form of data representation, but it can also be rather restrictive and can introduce special problems for the processing of subqueries. In general, the schema of a hierarchical database relates entities in a "tree structure" starting with a "root" node and branching for as many levels as are necessary to represent the application.

Any given node in the structure (except for the root node) must have a single parent. There can be as many children for a given parent as the application requires. This one-to-many relationship between parents and children (both node types and node occurrences) is quite useful for modeling a variety of real world situations that can be naturally viewed as hierarchies. Figure 2.3.1 shows a simplified example: the diagram above the dashed line represents the entity or node types (subschema) and the diagram below the dashed line depicts the node occurrences. Commercial hierarchical DBMSs include IBM's IMS and DL/1 and the SAS Institute's SYSTEM 2000. Numerous custom and military DBMSs employ tree-structured architectures.

Under the hierarchical data model a database record occurrence includes an occurrence of the root node and all occurrences of all dependent nodes. This concept of "record" presents some special problems for translation of the NDRL subquery to native format. The logical

A. Database Schema



B. Example Record Occurences

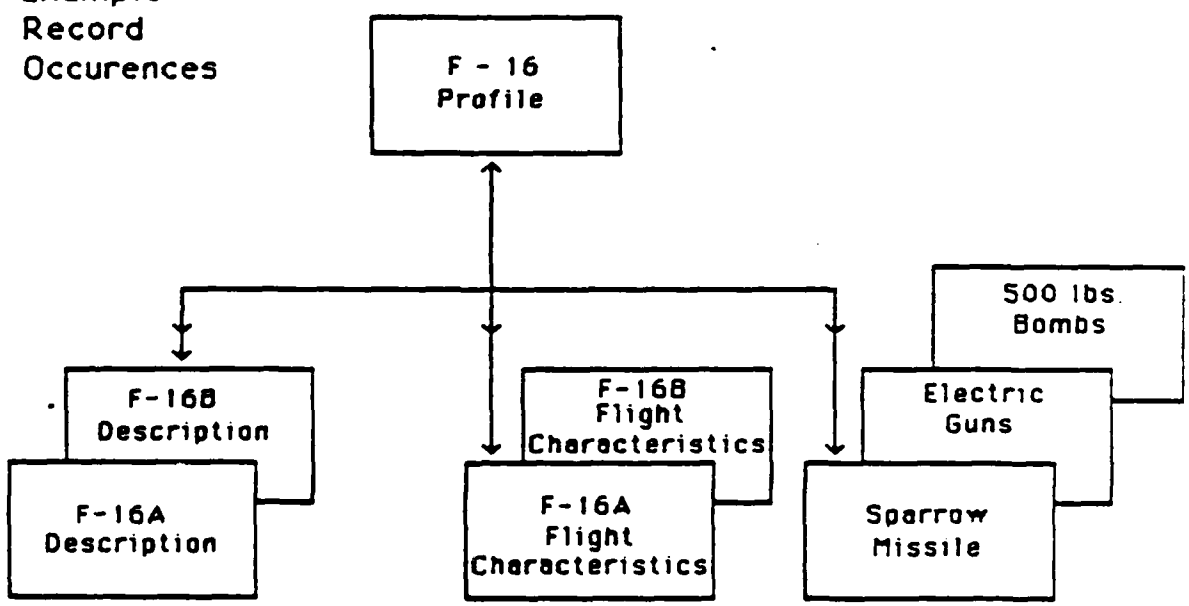


Figure 2.3.1

definition of Coherent Database Representations (CDRs) for hierarchical databases insures that subqueries which contain incomplete or ambiguous requirements will not be generated. Even so, the following cases may occur:

A. No root data element (primary key) is specified and the highest level data element specified (secondary key) has no restriction clause.

This condition should be trapped during the subquery decomposition process and an error returned to the user. If such a subquery does get through to the D1* Node, an error message is generated and the subquery is terminated. This type of subquery cannot be processed since it implied an "ALL" and could result in an entire database being selected by the subquery.

B. A root data element is specified, but no restriction for this primary key is given.

This condition must be disallowed. Although the key is explicit, such a subquery could result in a voluminous response. If such a subquery is received at the D1*, an error message must be generated, and if the volume exceeds the user's quota, the subquery must be terminated.

C. Data elements are selected from a subordinate node and for an ancestor (e.g., "grandparent" node), but not for the immediate parent node.

This condition should also be trapped by the subquery decomposition process since it makes the query ambiguous. If such a subquery is received at the D1* Node then an error message is generated. This imposes a requirement for "hierarchical continuity" upon subqueries directed at a hierarchical data model database.

D. The primary key and / or a secondary key is specified and restricted in value, and data elements are specified from contiguous subordinate nodes with or without restrictions on their values.

This is the general case for a valid subquery against a hierarchical database. The CDR representations of "records" for hierarchical databases reflects this requirement and the DL* enforces it as part of the process of translation of the subquery to the NDRL.

Although the particular syntax for interrogating a hierarchical data model database through a facility of its DBMS may vary considerably from one hierarchical data model DBMS to another, a subquery in NDRL format should require relatively minor syntactic changes and extensions in order to process it correctly. The specific syntax of the native query depends upon the particular DBMS, but such changes from the NDRL to native format should conform to the general description of NDRL transformations given in Section 2.3.1.

As discussed in Section 2.3.1, if a given DBMS does not provide a high level language or facility for accessing its databases, it may be necessary to generate a procedural program using primitive operations such as "FIND" and "GET" which can provide the data required by the subquery. If this is the case, then the program will have to establish and maintain "currency" within the target database. Currency is analogous to a set of "cursors" or "pointers" within the database that are used by a program to determine which records and entities are currently being accessed. Currency is usually established by first selecting a root node occurrence and then selecting a particular occurrence of each subordinate node until a particular node occurrence has been accessed at the lowest level required by the query. Usually all occurrences are searched to determine the lower level node type associated with a given occurrence of a superior node before the next occurrence of the superior node type is accessed. In this manner, a systematic traversal of the hierarchy or tree structure is possible.

2.3.3 Subquery Processing Against the Network Data Model

The network data model represents an increase in complexity and sophistication over the hierarchical model. With the network approach any given entity may have two or more parents rather than the single parent permitted in the hierarchical case. Thus the network model is a superset of the hierarchical model. One implementation of the network approach is the CODASYL (Committee on Data System Languages) standard DBMS. There are also other implementations which vary considerably from the CODASYL approach.

For the network model the concept of a "set" of entities or records is introduced; the set is used to associate one entity or record type with another. The superior record type is referred to as the "owner" of the set while one or more inferior record types are considered the "members". A given record type can participate in any number of sets either as the owner or a member. The only general restriction is that the same record type cannot participate as both the owner and a member of the same set. This restriction eliminates the possibility of direct representations of "recursive" data structures (such as a parts explosion structure) and often causes the introduction of "artificial" or "linkage" record types in order to represent the desired data relationships. The restriction may be relaxed in some network DBMS implementations.

The network model is sufficiently rich that it has found application to a wide variety of data base problems. The data structures can become rather complex and are rather rigid so considerable analysis is required before a commitment is made to a particular database design. Figure 2.3.2 shows both a subschema diagram and an entity / record occurrence diagram for a simple network database. Such a database could be implemented using any one of a number of commercial and military DBMSs including Cullinet's IDMS.

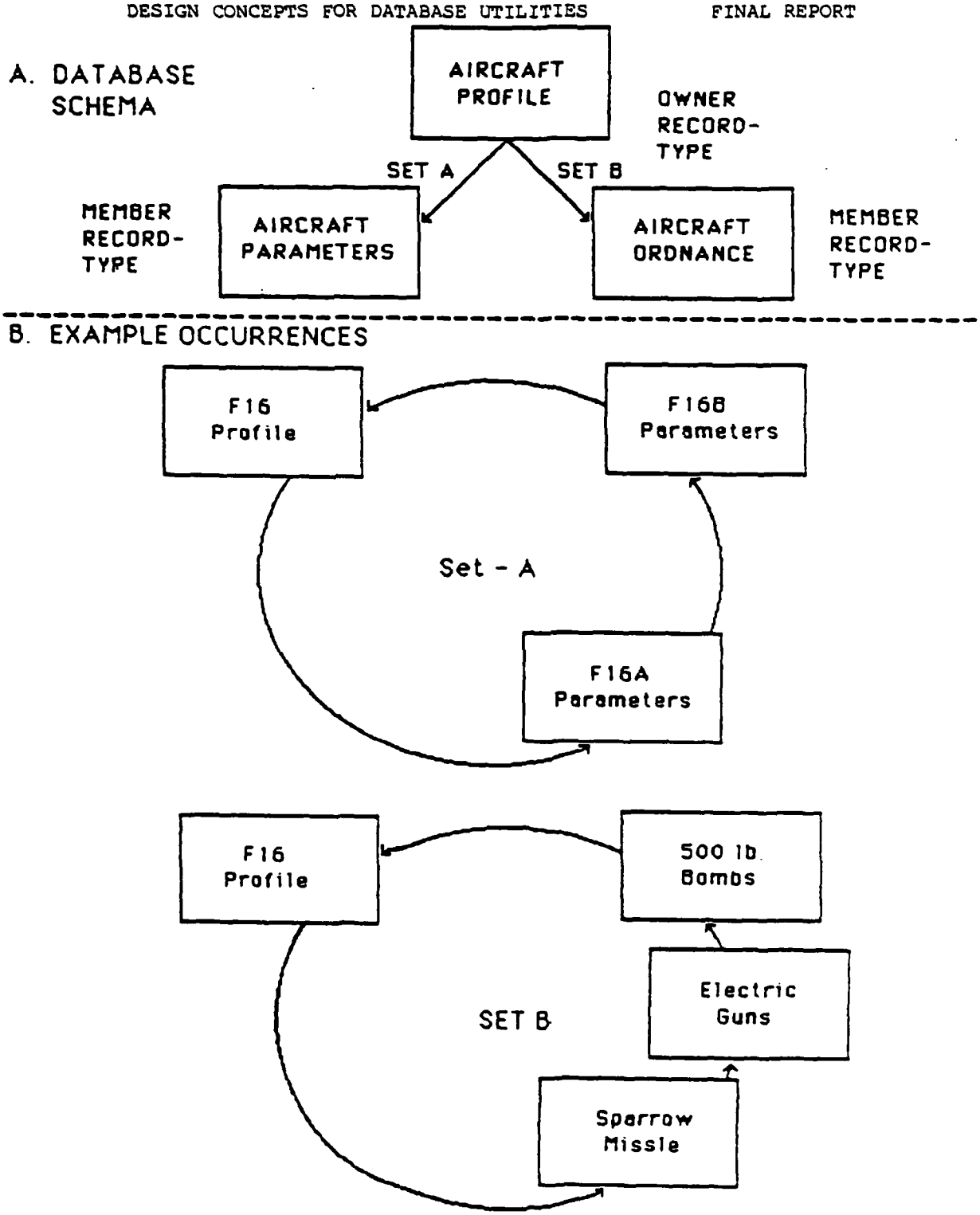

A. DATABASE SCHEMA
AIRCRAFT PROFILE
OWNER RECORD-TYPE
SET A
SET B
MEMBER RECORD-TYPE
AIRCRAFT PARAMETERS
AIRCRAFT ORDNANCE
MEMBER RECORD-TYPE
B. EXAMPLE OCCURRENCES
F16 Profile
F16B Parameters
Set - A
F16A Parameters
F16 Profile
500 lb Bombs
Electric Guns
SET B
Sparrow Missle

Figure 2.3.2

In addition to the basic concepts of sets, owners, and members, network databases may employ other concepts as well. A given database may be partitioned into different "Areas" in order to control access to certain data or improve processing performance. The database may exist as one or more physical files on one or more secondary storage devices. Areas are mapped to one or more files. A given data record can be located within a particular page in a given area in one of several ways. These record storage and retrieval "location modes" typically include:

- o Calc - a hash function is used to select a page.
- o Via - a record is stored as physically close to another record as possible.
- o Direct - a record is placed on a particular, user-specified, page.
- o Sequential- a record is stored in sequential order based upon a key value.

In addition to the above modes for locating a record, the DBMS may also provide an "indexed" or B-tree access method. Inverted lists or balanced trees are constructed for particular data element types within some of the record types in the database. These secondary keys permit fast access to groups of records within the database.

The translation of subqueries from the NDRL to native format for network DBMSs must consider a variety of conditions that are unique to this data model. In general, these conditions are anticipated when defining records in the CDR for the network databases and are, therefore, taken into account by the query decomposition process. The following cases must be considered:

A. Data elements are specified for a member record type, but no owner record type is specified and no value restriction is included.

This type of subquery cannot be processed since the volume of the subresponse could be large and the subquery is ambiguous.

B. Both an owner and one or more member records are included, but no value restriction is placed upon the primary key within the owner record.

This case can be processed but an error condition will result if the volume of the subresponse exceeds the user's quota.

C. Data elements are specified from record types which are not directly associated by owner-member relationships.

This case should be eliminated by virtue of the CDR representations of the network databases and the decomposition of subqueries to rather simple subqueries. If such a subquery should occur, it cannot be processed and would produce an error response. Such a subquery would be extremely ambiguous since there would be no direct way to establish relationships among the records selected.

D. The primary key for an owner record type is specified along with a restriction for its value, and one or more member record types are specified with or without value restrictions.

This is the usual case for subqueries that are processed against network databases.

The translation of subqueries from the NDRL to native format for network DBMSs must consider all of the points reviewed in Section 2.3.1. In addition, the native format subquery may include:

- A declaration of the Area to be interrogated.

- A specification of the file(s) to be searched.
- A specification of the location mode or retrieval method.
- A declaration of the subschema used for the retrieval.
- A listing of the sets to be searched.
- A specification of the record name(s) included in set.
- The qualification of data element names by record names.

How much or how little of the above must be included in the subquery depends upon the particular DBMS and the nature of the facilities it provides for query and report generation. All information that is needed for the translation is maintained in the subschema at the D1* and is included with the NDRL subquery that is sent to the Data Node.

If the local DBMS does not provide a high level query or report writer facility, it will be necessary for the IDN interface at the Data Node to translate the NDRL subquery into a procedural program in order to retrieve the desired information. Such a program would make use of primitive operators like FIND, GET, and OBTAIN to locate and extract the data. Member records would have to be processed within each set for each qualified owner record. Control of the retrieval process would be maintained using "currency indicators" like the following:

- Current of run unit - the most recently accessed record occurrence.
- Current of record type - the most recently accessed record occurrence of each record type.
- Current of set - the most recently accessed record occurrence within each set.

- Current of area - the most recently accessed record occurrence within each area.

Most modern network DBMSs provide retrieval facilities which mask these details of data retrieval and minimize the knowledge required above the database that is required. Such facilities are adequate to handle most simple to moderately complex queries. Since the subqueries generated by the IDN are rather elementary (due to the nature of the decomposition process), it should be possible to process them through high level facilities and thus eliminate the need to generate complex retrieval programs.

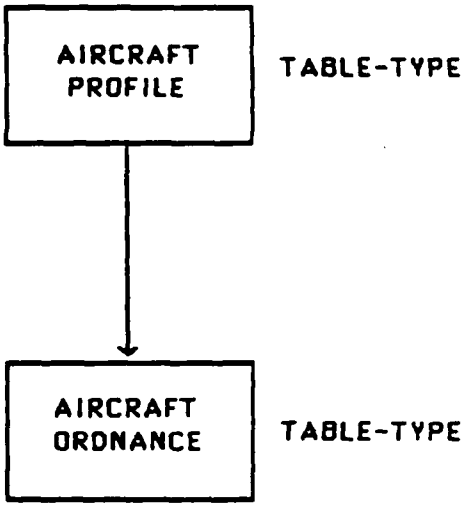
2.3.4 Subquery Processing Against the Relational Data Model

The relational data model is the most simple and straightforward of the data models reviewed thus far. Relational databases consist of relations (a tabular form) with unique primary keys and zero, one, or many associated attributes (data element). Each row within a relation is considered to be a record. Relational databases are becoming increasingly popular and are represented by such commercial products as IBM's SQL/DS. Figure 2.3.3 provides an entity diagram and an occurrence diagram for a simple relational data model database.

The relations within a relational database are associated or related to one another by common attributes. A primary key in one relation may appear as a non-primary-key data element type or "foreign key" in another. By performing JOIN operations based upon common attribute types, it is possible to retrieve related information from two or more relations.

In addition, most relational DBMSs provide capabilities for maintaining secondary indices or B-trees for individual attributes. These auxiliary

A. DATABASE SCHEMA



B. RECORD OCCURRENCES

AC-Type	Aircraft Class	Wgt.	Flight Parameters					
F-16A	Fighter/Bomber	12,000	—	—	—	—	—	—
F-16B	Fighter/Bomber	13,000	—	—	—	—	—	—

AC-Type	Ordnance Type	Supply Code	Load	Wgt.	Range
F16A	Sparrow	—	—	—	—
F16B	Guns	—	—	—	—

Figure 2.3.3

files make it possible to quickly select records based upon one or more value restrictions. Once records have been selected by direct access based upon their primary key values, or by sequential scan, they can be related to other records or simply retrieved.

Another consideration for relational databases is the view concept. A logical view or "virtual relation" may be used to access data which is actually stored quite differently from the way it appears within a view. A view can be used to:

- o Mask data element types from a given relation.
- o Allow access to only a subset of a relation based upon a (predefined) selection criteria.
- o Present two or more (joined) relations as if they were a single relation.

Usually, the relational DBMS will permit views to be predefined and maintained in the internal dictionary of the relational database. Any subquery can, therefore, access a view in the same way it would access any relation.

The CDR representations for relational databases present each relation or view as a record type containing one or more data element. Only those views and relations which may be accessed by the IDN are given in the CDRs for the various data bases.

Execution of subqueries against the relational databases is relatively straightforward, especially since the subqueries are kept simple by the query decomposition process. Even for simple subqueries, however, the following cases must be considered:

A. No value restriction is provided for any data element in a given relation.

Such a subquery could result in the output of all records in the relation. An error condition will occur if the volume of the response exceeds the user's quota.

B. A join operation is required where one or both of the relations to be joined has not been subsetted through a value restriction.

This case could also result in considerable output as well as processing overhead. In general, this type of subquery will be considered to be an error. If the first relation to be joined has a restriction clause and an "= ALL" restriction clause is provided for the second relation, then the join operation will be permitted.

C. A value restriction clause is provided for a data element in each relation which participates in the subquery.

This is the usual case for the IDN subqueries against relational data model databases.

Section 2.3.1 provides an overview of the NDRL and its translation to the native language of the target database. Since the NDRL is very similar to most relational query languages (such as SQL), very little translation should be required to produce an executable subquery. The database subschemas maintained at the D1* provide any information which could not be included in the subqueries using the CDRs alone. This may include:

- o View to Record Mappings.
- o Physical file specifications.
- o Any predefined selection criteria which must be appended to subqueries to access specific relations.

Such information is included in the NDRL subquery to insure its compatibility with the DBMS and database at the target Data Node.

2.3.5 Subquery Processing Against the Inverted Model

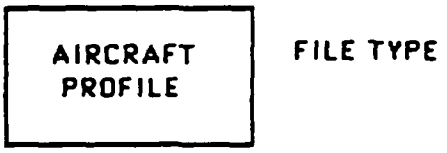
The inverted file architecture is so basic that it hardly deserves to be designated as a data model. Even so, an inverted file DBMS can provide a variety of powerful data retrieval capabilities. Many such systems mirror relational DBMS functionality to the point that it is difficult to distinguish one from another. There are numerous military and commercial inverted file systems; one of the most popular is Model 204 from Computer Corporation of America. Figure 2.3.4 provides a simple example of an inverted file database.

In this kind of database, each record type is associated with a file type. In order to relate two or more types of records, it is necessary to select data from the corresponding files. Access is made efficient by the extensive use of inverted lists or B-trees. There are often no limits placed on how many data element types within a given record type can be made "key", but performance and storage considerations usually limit the number of keys to 25% or less of the number of data element types in the record type.

Inverted file DBMSs always provide one or more high level ("English-like") query languages; hence, the translation of the NDRL subqueries to the native query language presents no major problem. Although the syntax of the target language may differ greatly from that of the NDRL, the basic concepts are the same.

Simple subqueries should map directly to simple queries against the inverted file database. If a subquery requires access to two or more files (record types), it may be necessary to generate a procedural program in order to execute the subquery successfully. This requirement depends on the nature of the query language provided by the DBMS. A more complex subquery may require the creation of temporary "selection sets" from one or more files and the procedural processing of the sets. Since the subquery decomposition process is designed to keep the

A. DATABASE SCHEMA



B. RECORD OCCURRENCES

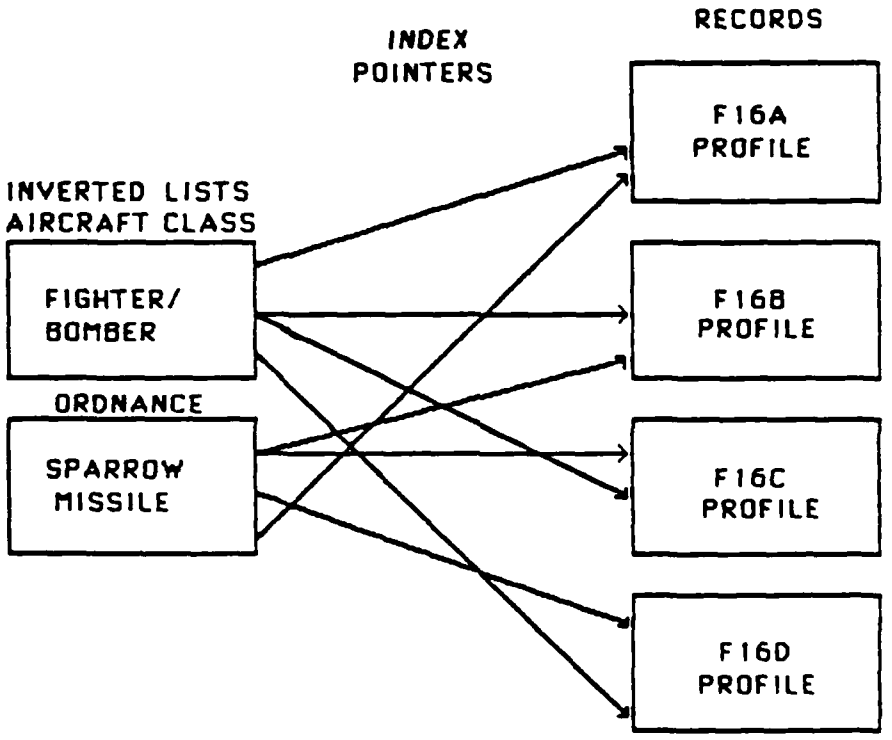


Figure 2.3.4

subqueries as simple as possible, the requirement to perform procedural processing should be rare.

As with all of the data models, there are specific cases which can arise during subquery processing which must be considered. These cases are:

- No restriction clause specified for any data element for a given record type.

  This case could result in the production of a large volume of output. An error condition will occur if the volume of data surpasses the user's quota.

- A restriction clause is specified for at least one data element for each record type which participates in the subquery.

  This is the usual case for subqueries which are to be processed against inverted file databases.

2.4 USING JOINED SUBSETS (SEMI-JOINs) IN SUBRESPONSE COMPOSITION

In order to minimize communication overhead while performing subresponse composition operations, it is important to develop a strategy, or algorithm, for performing these operations for a general set of subresponses. An approach to this problem is the "Wong algorithm" developed by E. Wong for CCA's SDD-1. His algorithm features the use of "joinable subsets" or "SEMI-JOINs" rather than "natural" JOINs. The natural JOIN matches relational rows in two relations based upon the full value domains of a common relational attribute. The SEMI-JOIN is, generally, a much more efficient procedure. In the following discussion of the SEMI-JOIN algorithm, the term JOIN is used to mean natural JOIN and the term attribute is used to mean relational attribute:

- Assume two subresponses X and Y have been extracted from two different databases and that X and Y have a common attribute J over which a JOIN may be performed.
- Assume that X resides at node A and Y resides at Node B.
- Compute the PROJECTION of Y over J at B and store this as T1.
- Send T1 to node A.
- Compute the JOIN of T1 and X over J at A and store this as T2.
- Send T2 to node B.
- Compute the JOIN of T2 and Y over J at B and store this as R1.

Here T2 is the SEMI-JOIN of X with respect to J and R1 is the natural JOIN of X and Y over J. The effect of this operation is the same as if a SELECTion had been performed on X using the values of J contained in Y and then the result JOINed with Y.

Wong shows that this algorithm will reduce the total data movement (which, for the IDN, is measured in bytes) if and only if:

(1) SIZE (T1) + SIZE (T2) < SIZE (X)

Otherwise, it is more efficient to simply move X from A to B and perform the JOIN. This operation eliminates the need to communicate those tuples which will not JOIN although it has the extra overhead of communicating the selection set T1. In computing the sizes of temporary files, it should be remembered that, in a JOIN operation, a single tuple in one relation may "match" more than one tuple in another relation; this has the result of increasing the size of the result over that of the initial relation.

It is also important to consider the size of the two original relations:

(2) IF SIZE (X) > SIZE (Y) THEN T1 = PROJECTION Y : J

ELSE IF SIZE (Y) > SIZE (X) THEN T1 = PROJECTION X : J

where "A:B" denotes "relation A over attribute B".

It is very important to note that calculation (1) cannot be computed without actually performing the PROJECTION and SEMI-JOIN operations; therefore an estimate of the value of this operation must be made based upon the computed sizes of X and Y. A commitment to using this approach means that the system must generally assume that the SEMI-JOIN will produce a smaller resultant subresponse (in bytes) than the size of

the smaller of the two original subresponses. Given the elementary nature of the subqueries, this assumption should be generally valid.

The implications of the SEMI-JOIN approach to subresponse composition for the entire assembly process are as follows:

- o The size of each subresponse must be known to the QUERY-MONITORING-AND-CONTROL Subsystem (QMC) before an "intelligent" decision can be made as to how SEMI-JOINs will take place and where the results should be assembled.
  - Even if size estimates for databases, record-types, and attribute-types are maintained with the CDRs, it is impossible to estimate in advance the size of any given subresponse (with variable selection criteria).
  - In order to determine over which subresponse the PROJECTION should be performed (i.e., which one is smaller), the size of each subresponse must be computed and forwarded to QMC after the subresponse has been generated.
  - If the size (in bytes) of one subresponse is much less than the size of another, say 10 - 20%, it is generally more efficient to send the smaller subresponse to the same site as the larger and then to perform a JOIN.
- o In order to minimize communication of irrelevant records among nodes, it is desirable to be able to perform SEMI-JOINs at the D1* Nodes as well as the D2Q Nodes. If these operations can only be performed at D2Q Nodes, then QMC gains no significant advantage by sending subresponses to the same D2Q Node for assembly.
- o If this procedure is applied successively to multiple subresponses (i.e., if X SEMI-JOIN Y --> R1 and R1 SEMI-JOIN Z --> R2, etc.)

then, in order to optimize the computation of the result, it would be advantageous to know the sizes of all of the subresponses in advance before any of the SEMI-JOINs are performed.

CHAPTER 3

SUBSYSTEMS SPECIFICATION

This chapter contains the specification of the fourteen IDN subsystems that are required for the processing of queries.

The detailed description of the data flows and the schema of IDN dictionaries, which are referenced, are given in Appendices I and II of this report.

(This page intentionally left blank)

3.1 SUBSYSTEM 1: PROVIDE-LOCAL-DD-SUPPORT

3.1.1 OVERVIEW

Subsystem 1, the PROVIDE-LOCAL-DD-SUPPORT Subsystem is a special front-end to the IRDS resident at the D1 Node. This Subsystem provides all support facilities for users who specify or run IDN queries. These facilities consist of:

- Storing and Retrieving IDN Queries.
- Obtaining a listing of all or selected queries for a particular user.
- Obtaining the listing of all or selected data element names which are within the user's Local User Name Space (LUNS).
- Obtaining a cross-reference of data elements to queries.
- Obtaining information on any or all functions.

A user's view of the dictionary is restricted so that:

- The user is not able to obtain from the dictionary any information about security-related descriptors (VIEW and DICTIONARY-USER entities and DICTIONARY-USER-HAS-VIEW relationships).
- The user can obtain information only about the user's LUNS or queries.

- The user cannot obtain the name of the CDR-ELEMENT corresponding to any LOCAL-ELEMENT-NAME. The user's perception of an element consists of the LOCAL-ELEMENT-NAME and the properties of the corresponding CDR-ELEMENT.

The D1 Node Administrator has access to all the IRDS dictionary commands which underlie this Subsystem.

3.1.2 INTERFACE SPECIFICATIONS

3.1.2.1 Intersubsystem Data Flows

Figure 3.1.1 shows the data flows between this and other Subsystems.

3.1.2.2 Subsystem Events

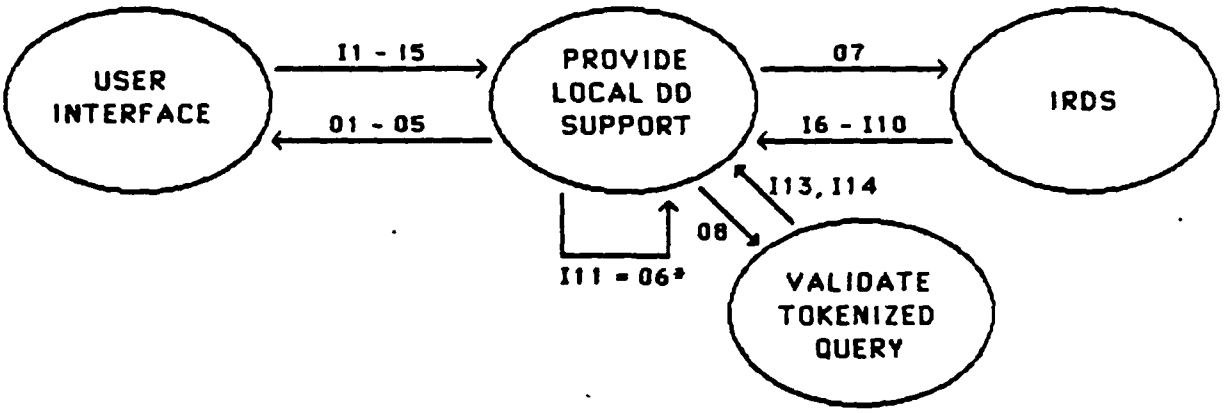
Does not apply.

3.1.3 TECHNICAL REQUIREMENTS

3.1.3.1 Actions

This Subsystem filters dictionary transactions. The dictionary transaction may come either from a direct user request, or from some facility of the user interface. (For example, a request for help on FUNCTIONs might trigger a request for a listing of all FUNCTIONs.) The dictionary transaction may either be a simple, standard IRDS transaction, or a custom transaction.

Subsystem 1: PROVIDE-LOCAL-DD-SUPPORT



In	Data Item	Source Subsystem
I1	Save Query Transaction	User-Interface
I2	Delete Query Transaction	User-Interface
I3	Retrieve Query Transaction	User-Interface
I4	Report Dictionary Transaction	User-Interface
I5	IRDS Native Mode Command	User-Interface
I6	Check Query Transaction	User-Interface
I7	Standard IRDS Output	IRDS
I8	Custom IRDS Output	IRDS
I9	IRDS Transaction Confirmation	IRDS
I10	IRDS Transaction Diagnostics	IRDS
I11*	Stored Query	IRDS
I12	IRDS Native Mode Maintenance Command (internal format)	PROVIDE-LOCAL-DD-SUPPORT-AT-PRIMARY-D1
I13	Intermediate Query	VALIDATE-TOKENIZED-QUERY
I14	Query Diagnostics	VALIDATE-TOKENIZED-QUERY

Out	Data Item	Destination Subsystem
O1	Transaction Confirmation	User-Interface
O2	Transaction Diagnostics	User-Interface
O3	Stored Query Content	User-Interface
O4	Custom Report Output	User-Interface
O5	Native IRDS Output	User-Interface
O6*	IRDS Native Mode Maintenance Command (internal format) IRDS Native Mode Command (internal format)	PROVIDE-LOCAL-DD-SUPPORT-AT-ALTERNATE-D1
O7	IRDS Native Mode Command (internal format)	IRDS
O8	Check Query Transaction	VALIDATE-QUERY-TRANSACTION

* NOTE: This data flow exists between the primary and alternate D1's. It is an output from the primary D1 and an input at the alternate D1.

Figure 3.1.1

The standard transactions include:

- o All dictionary maintenance commands except for Modify Entity Life-Cycle Phase (Section 5.2.1 of the dpANS IRDS);
- o All dictionary output commands (Section 5.2.2 of the dpANS IRDS);

(There is no need for the Modify Entity Life Cycle Phase command in this environment. Also, there is little need in this environment for entity-list and dictionary procedure commands, Sections 5.2.3 and 5.2.4 of the dpANS IRDS, respectively. The schema commands are not available to any user of the D1 dictionary.)

The transaction is then processed by the appropriate transaction function. Each transaction process generates one or more primitive dictionary transactions which are executed via the internal dictionary access / update call interface. The results of the internal transactions are interpreted and all output is formatted for return to the user interface. All update transactions are trapped and routed to the PROVIDE LOCAL DD SUPPORT Subsystem at each shadow D1 node.

Only the D1 Node Administrator will have access to the standard IRDS dictionary command transactions. All other transactions will be customized.

3.1.3.2 Intrasubsystem Data Flows

The intrasubsystem data flows are defined in Figure 3.1.2.

Subsystem 1: PROVIDE-LOCAL-DD-SUPPORT

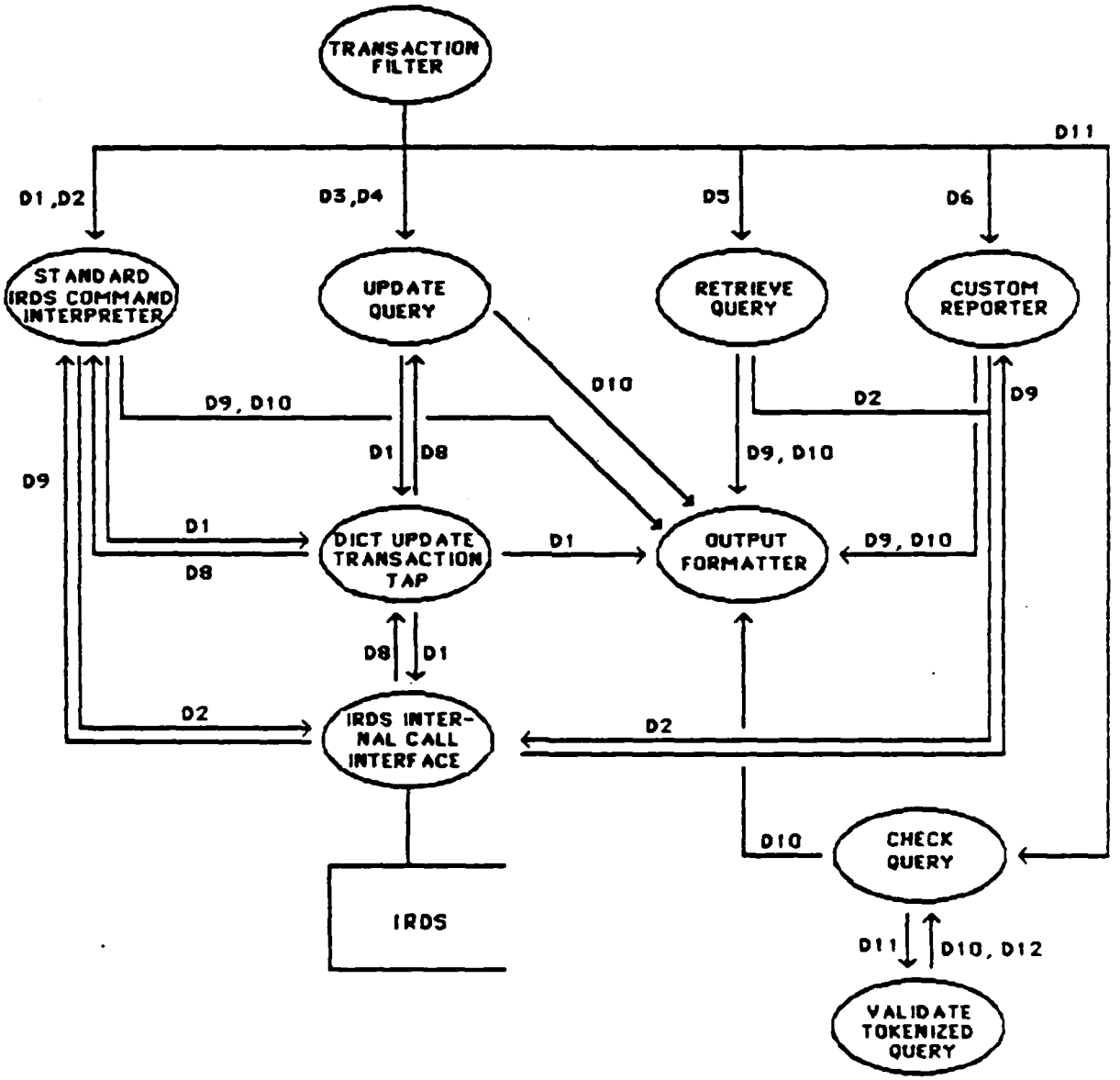


Figure 3.1.2

Subsystem 1: PROVIDE-LOCAL-DD-SUPPORT

Data Flow		Source Function	Destination Function
D1	dpANS dictionary update command	Transaction Filter	Standard Command Interpreter
		Standard Command Interpreter	Dict Update Transaction Tap
		Update Query	Dict Update Transaction Tap
		Dict Update Transaction Tap	IRDS Internal Call Interface
		Dict Update Transaction Tap	Output Formatter
D2	dpANS dictionary output command	Transaction Filter	Standard Command Interpreter
		Standard Command Interpreter	IRDS Internal Call Interface
		Retrieve Query	IRDS Internal Call Interface
		Custom Reporter	IRDS Internal Call Interface
D3	save-query transaction	Transaction Filter	Update Query
D4	delete-query-transaction	Transaction Filter	Update Query
D5	retrieve-query transaction	Transaction Filter	Retrieve Query
D6	LIST transaction	Transaction Filter	Custom Reporter
D7	SHOW transaction	Transaction Filter	Custom Reporter
D8	return code	IRDS Internal Call Interface	Dict Update Transaction Tap
		Dict Update Transaction Tap	Update Query
		Dict Update Transaction Tap	Standard Command Interpreter
D9	internal form of dictionary output	Retrieve Query	Output Formatter
		Custom Reporter	Output Formatter
D10	diagnostic messages	Standard Command Interpreter	Output Formatter
		Standard Command Interpreter	Output Formatter
		Update Query	Output Formatter
		Retrieve Query	Output Formatter
		Custom Reporter	Output Formatter
		Check Query	Output Formatter
D11	tokenized query	Transaction Filter	Check-Query
		Check-Query	VALIDATE-TOKENIZED-QUERY
D12	intermediate query	VALIDATE-TOKENIZED-QUERY	Check-Query

Figure 3.1.2

3.1.3.3 Tasking

Below the IRDS Internal Call Interface, the IRDS will be composed of one or more tasks. The transaction filter will be a single task. Each transaction will have a task associated to process it. A single task will be dedicated to output.

3.1.3.4 Exception Conditions .

None.

3.1.3.5 Rationale for Critical Decisions

1. Relationships between STORED-QUERY and LOCAL-ELEMENT-NAME entities are maintained.

   This provides the user with several benefits:

   a. It enables a user to browse the dictionary to find previously defined queries which may be the basis for newly defined queries. It also may help avoid generating duplicate queries.

   b. It enables a new user to become more familiar with certain data elements in his LUNS by identifying queries which use them and seeing how these data elements are used.

   It also provides the D1 administrator with several benefits:

   a. As the data definitions evolve over time, it enables the D1 Node Administrator to determine potentially effected queries and notify users of the need for them to modify the queries accordingly.

b. It gives the D1 Node Administrator a better view of the usage of data elements by different user communities. In order to store a query and establish relationships from the stored QUERY entity to each of the referenced LOCAL-ELEMENT-NAMEs, it is necessary to validate the tokenized-query according to the rules of Subsystem 2. Relationships between the STORED-QUERY and LOCAL-ELEMENT-NAMES are established when the STORED-QUERY is validated.

This approach permits the user to save incomplete or partially correct queries. However, it means that for a validated query, either both the tokenized and intermediate query must be saved, or the user-interface must "decompile" a intermediate query when changes are made to the tokenized query. The former approach is chosen because of its simplicity.

2. Both validated and non-validated queries may be stored.

If the user were only able to store validated queries, partially completed queries could not be saved. This would be very unfriendly.

3.1.3.6 Open / Deferred Decisions

1. Should multiple generations of queries be supported? If so, should the incrementing of generation-numbers be under user control or system control?

The IRDS supports version control. Maintenance of multiple generations of queries could provide the user with some benefits. On the other hand, it will introduce some complications into the user interface.

3.1.3.7 Critical Algorithms

None.

3.1.4 DATA REQUIREMENTS

3.1.4.1 Explanation of Major Data Flows

Save-query-transaction

This transaction saves a query in the D1 dictionary. If the query has passed Subsystem 2 validation, then relationships between LOCAL-ELEMENT-NAMEs and the STORED-QUERY are also established. The saved query is also shipped to all of the shadow D1 nodes for saving. If multiple generations are not used for queries, then a save query transaction will replace the existing query. Otherwise, if the number of generations of the query exceeds an installation defined limit, it will delete the oldest generation.

Delete-query-transaction

This transaction deletes the STORED-QUERY entity and all its relationships to LOCAL-ELEMENT-NAMEs.

Retrieve-query-transaction

This transaction retrieves a stored query transaction from the D1 IRDS, and returns it to the user-interface for modification or execution.

Check-query-transaction

This transaction causes the query to be validated according to the

rules defined in Subsystem 2. If the query passes this validation, the PROVIDE-LOCAL-DD-SUPPORT Subsystem receives the intermediate form of the query, which the user may then save.

Customized Output Transactions

The IRDS reporting and querying facilities available to the IDN query-user are a subset of the full dpANS IRDS output facilities. Since the IDN query-user's view of the IRDS is very simple, the IRDS querying and reporting facilities are also very simple. Two specialized IRDS output transactions are provided:

- The LIST-Transaction, which provides a simple list of names of selected entities; and
- The SHOW-Transaction, which provides detail information about selected entities.

We use an abstract syntax to demonstrate the options available to the query user for reporting on IRDS contents. The specification of this syntax does not imply that a user will use this (or any) syntax in formulating requests for IRDS output.

LIST-Transaction

```
LIST  { QUERY | ELEMENT | FUNCTION }
[ scan-mask-1, scan-mask-2, ... scan-mask-n ]
[ WHERE restriction-criteria ]
[ ROUTE destination-id ]
```

LIST provides a list of entity names in name sequence. If no scan-mask is specified, then all entities, of the specified type, visible to the user are listed. Restriction criteria may be used to further qualify selection, and output may be routed to a specified destination (presumably a local printer or the user's terminal.)

Common Capabilities for LIST and SHOW

Use of Wild Cards

For both commands, the optional scan-masks may either be entity names, or character strings with "wild card" characters. An asterisk (*) indicates any sequence of characters, and a question-mark (?) indicates any single character.

Restriction Criteria

The optional restriction-criteria consist of boolean combinations of tests of entity properties, or a "related to" criterion. The boolean operators AND, OR, and NOT are supported, and parentheses control the nesting of criteria.

SHOW-Transaction

```
SHOW { QUERY | ELEMENT | FUNCTION }
[ scan-mask-1, scan-mask-2, ... scan-mask-n ]
[ WHERE restriction-criteria ]
[ ROUTE destination-id ]
[ SORT sequence-arg-1 [ ... sequence-arg-n ] ]
[ DISPLAY display-option-1 [ ... display-option-n ] ]
```

SHOW provides detail information for selected entities. Like LIST, if no scan-mask is specified, then all entities of the specified type visible to the user are selected. Restriction criteria may also be used to further qualify selection. Output may be routed to a specified destination.

Unlike LIST, which displays only entity names, display options are available. The display options enable the user to show all or selected properties, relationships, and, in the case of

a QUERY, the syntax and/or resultant form. SHOW also lets the user of query control the sequencing of output for each entity.

Special Options for SHOW Only

Sequencing

The sequence-arguments are ordered pairs in parentheses. The first component of the pair is either NAME, or the name of a property-type for the particular entity-type. The second component is an ASCENDING/DESCENDING indicator. The series of sequence arguments identify major to minor sort key. If no SORT parameter is specified, the sequence option of "(NAME,ASCENDING)" is assumed.

Display Options

Minimal display options are: DISPLAY ALL, DISPLAY PROPERTIES, DISPLAY RELATIONSHIPS, DISPLAY SYNTAX (for queries only) and DISPLAY specific property. If no DISPLAY parameter is specified, ALL is assumed.

Standard IRDS Command Transactions

The following IRDS maintenance commands are supported:

- Add entity
- Modify entity
- Delete entity
- Add relationship
- Modify relationship
- Delete relationship
- Modify access-name
- Modify descriptive-name
- Copy entity

The following IRDS output commands are supported:

- o General Output Command
- o Output Impact-of-Change

In both these cases, the options for using entity-lists are not supported.

3.1.4.2 *Special Data Requirements*

Because some transactions (save query, delete query) require execution of multiple dictionary update commands, it is necessary that the IRDS internal call interface be able to "batch" multiple commands.

3.1.5 DATABASE REQUIREMENTS

3.1.5.1 Network Metadata

This Subsystem must know each D1 node which can act as a shadow D1 (D1/S).

3.1.5.2 Data Metadata

The D1 IRDS View (see Figure 3.1.3) shows the types of entities and relationships actually held in the D1 dictionary. However, the query user's perception of the D1 dictionary is much more restricted. To the query user, the dictionary contains the following types of descriptors:

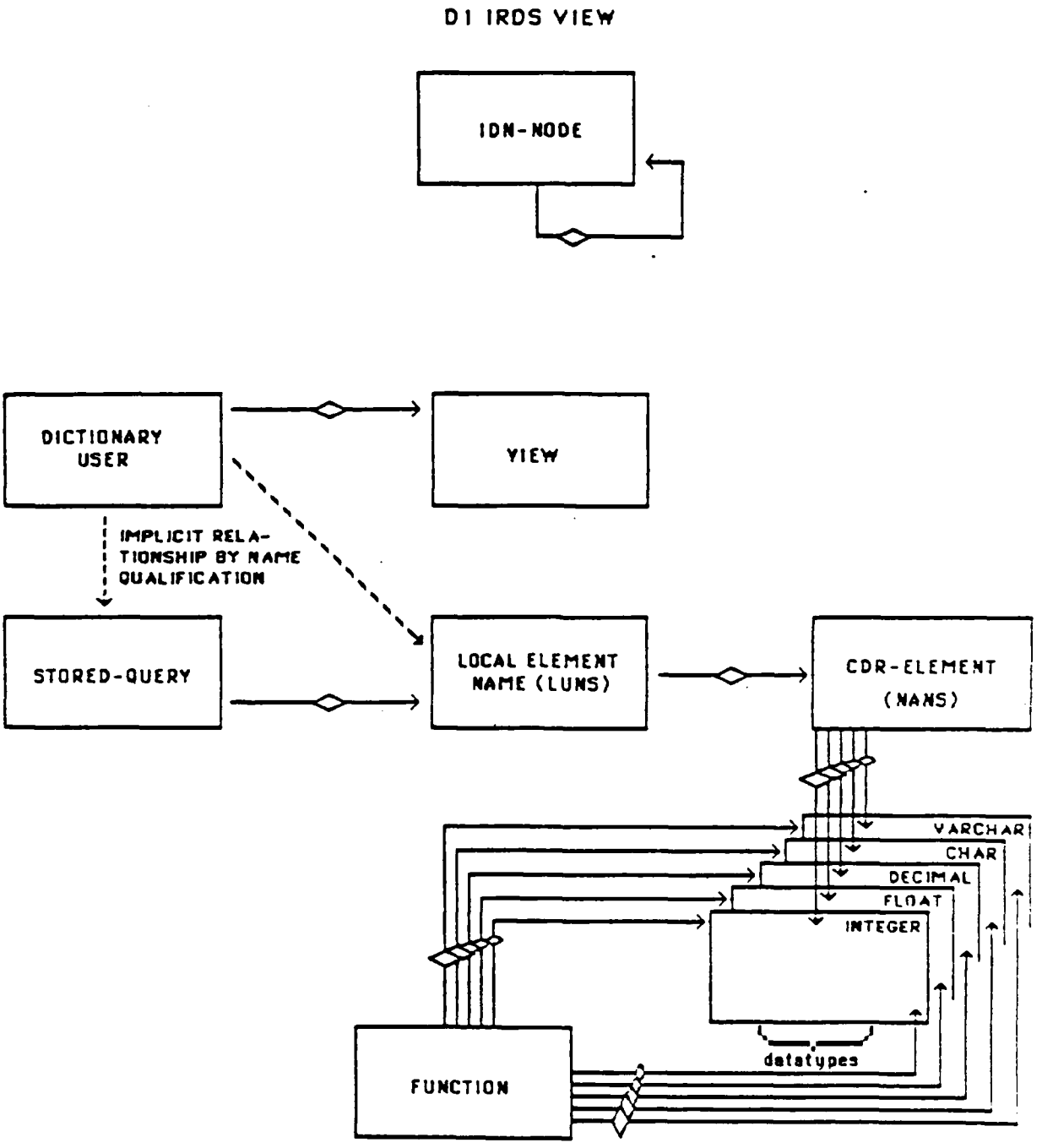

D1 IRDS VIEW
IDN-NODE
DICTIONARY USER
VIEW
IMPLICIT RELA-TIONSHIP BY NAME QUALIFICATION
STORED-QUERY
LOCAL ELEMENT NAME (LUNS)
CDR-ELEMENT (NAMS)
VARCHAR
CHAR
DECIMAL
FLOAT
INTEGER
datatypes
FUNCTION

Figure 3.1.3

- "Entity-types"

  STORED-QUERY
  FUNCTION
  ELEMENT

- "Relationship-types"

  QUERY-USES-ELEMENT

For the query user, an ELEMENT may be perceived as combination of the assigned-access-name of a LOCAL-ELEMENT-NAME entity and the properties of the corresponding CDR-ELEMENT. The query user is not aware of the name of the CDR-ELEMENT, and, as far as he is concerned, "element-name REPRESENTED-AS data-type-name" is a property-group rather than a relationship. Also, the query user is not aware of any version qualifiers as part of the entity names. This Subsystem automatically substitutes the user's identifier for the variation-name in the access-name for STORED-QUERY and LOCAL-ELEMENT-NAMEs. In effect, this partitions the STORED-QUERY and LOCAL-ELEMENT-NAME entities by user. The query user has no access to STORED-QUERY and LOCAL-ELEMENT-NAME entities which do not have his user identifier in the variation-name.

3.1.5.3 Other

None.

3.1.6 NOTES

1. The user-interface should also provide a Query-status transaction.

(This page left intentionally blank)

3.2 SUBSYSTEM 2: VALIDATE-TOKENIZED-QUERY

3.2.1 OVERVIEW

Subsystem 2, the VALIDATE-TOKENIZED-QUERY Subsystem, accepts a query in the IDN-standard form which is produced by the user-interface (the tokenized-query). It starts the process of building a Query Control Block (QCB) and transforming the query into canonical form. The form of the query which is output by this Subsystem is called the intermediate query form. The transformation of the query from the form produced by the user interface into canonical form is accomplished in several phases by different Subsystems.

This Subsystem performs that syntactic and semantic validation which can be performed at a DI Node. This validation includes:

- Identification of duplicate form-element declarations. (See Section 3.2.4.1)
- Identification of missing form-element declarations. (See Section 3.2.4.1)
- Identification of incorrect usage of names, such as
  - using a LOCAL-USER-NAME as a function;
  - using a function-name as a form-element-name.
- Identification of data type conflicts in expressions and function arguments.
- Identification of formatting errors.

- Identification of incorrect usage of an operator.

This Subsystem also interfaces with the PROVIDE-LOCAL-DD-SUPPORT Subsystem in order to process the Check-query-transaction. Validation is the same, but all diagnostics are returned to the PROVIDE-LOCAL-DD-SUPPORT Subsystem.

3.2.2 INTERFACE SPECIFICATIONS

3.2.2.1 Intersubsystem Data Flow Diagram

Figure 3.2.1 shows the data flows between this and other Subsystems.

3.2.2.2 Subsystem Events

The event diagram for this Subsystem is shown in Figure 3.2.2.

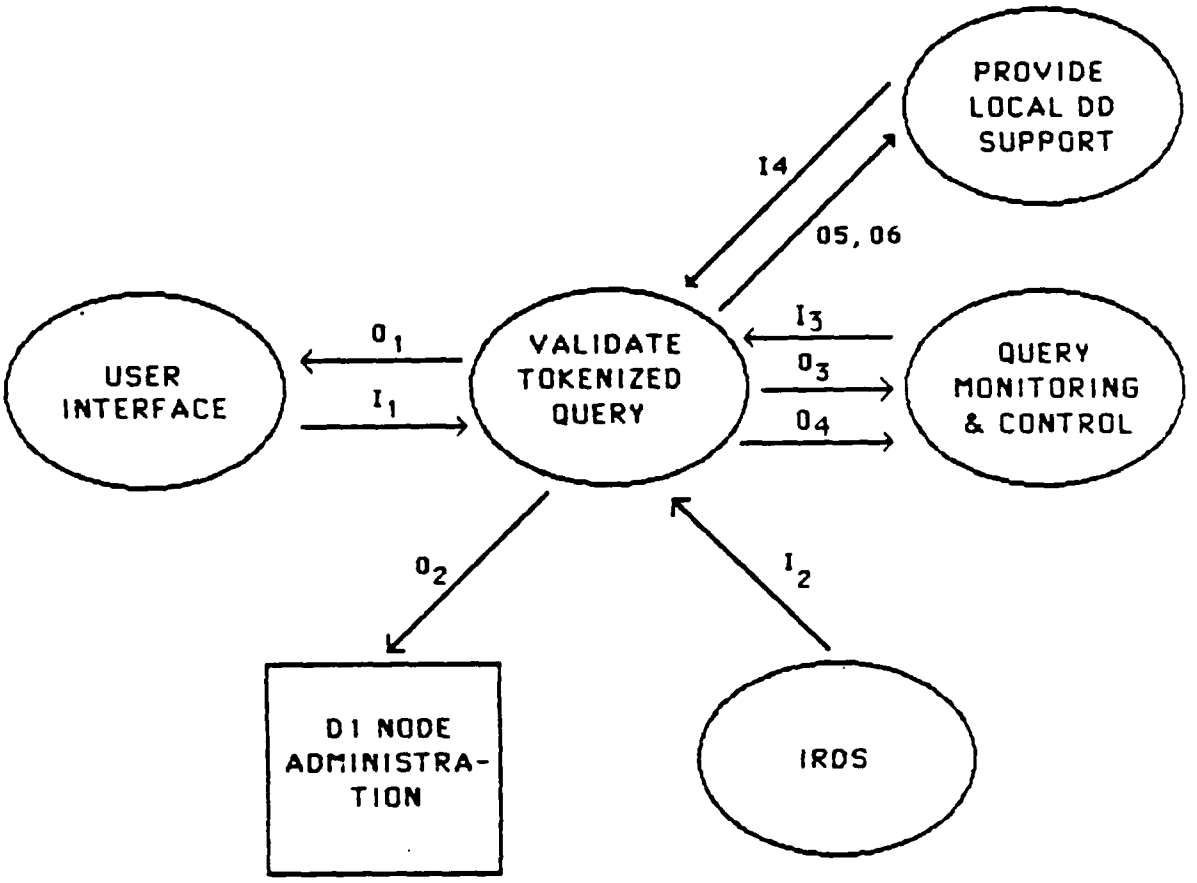
3.2.3 TECHNICAL REQUIREMENTS

3.2.3.1 Actions

Upon receipt of the tokenized query, a task is established to perform initial semantic validation. The task generates a Query Control Block (QCB) header for the query.

As the tokenized query is validated, it is transformed into intermediate form. The intermediate form of the query includes a symbol table. Each label in the tokenized-query is placed in the symbol table. Duplicate

Subsystem 2: VALIDATE-TOKENIZED-QUERY

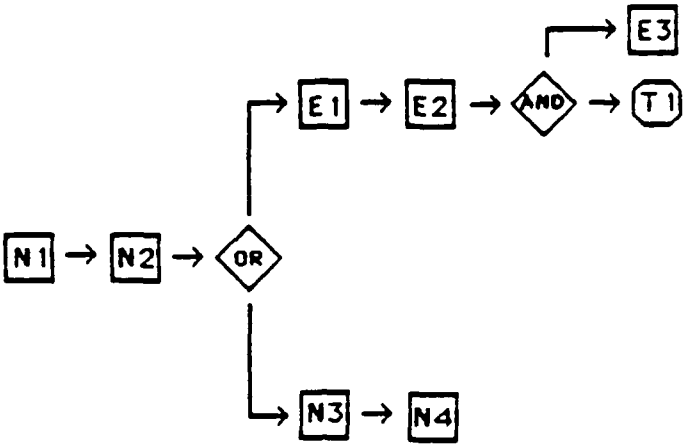


In	Data Item	Source
I1	Tokenized-Query	User-Node
I2	Local User Name Space	IRDS
I3	(LUNS)	QUERY-MONITORING-AND-CONTROL
I4	Destination-Data	PROVIDE-LOCAL-DD-SUPPORT
	Check-query-transaction	

Out	Data Item	Destination
01	Query-Diagnostic	User-node
02	Admin-Message	D1-node-administrator
03	Intermediate-query	QUERY-MONITORING-AND-CONTROL
04	Destination-Request	QUERY-MONITORING-AND-CONTROL
05	Diagnostics Messages	PROVIDE-LOCAL-DD-SUPPORT
06	Intermediate-Query	PROVIDE-LOCAL-DD-SUPPORT

Figure 3.2.1

Subsystem 2: VALIDATE-TOKENIZED-QUERY



Normal	Event Description
N1	Tokenized Query is Input at D1 node
N2	Validation Task is Inititiated
N3	Query Passes Validation
N4	Intermediate-query Passed to Query Monitoring & Control

Error	Event Description
E1	Query Fails Validation
E2	Diagnostics Output
E3	User Interface Receives Notification of Query Failure

Terminal	Event Description
T1	Query Terminated

Figure 3.2.2

declarations are identified. As each declaration, clause, or expression is scanned for labels, each label is replaced by a reference to the corresponding symbol table entry.

Each label is checked against the D1 dictionary to determine if it is:

- A LOCAL-ELEMENT-NAME for a CDR-ELEMENT;
- A FUNCTION;
- A computational result; or
- Invalid.

Inconsistent usages and data typing conflicts are identified, and diagnostic messages are generated.

All diagnostic messages are placed in a queue. If the query was stored and previously run successfully, but failed this level of validation, an administrative message is produced. This message identifies that a change in the dictionary has invalidated this stored query.

If the query fails this level of validation, all diagnostic messages are output to appropriate destinations and the task is terminated.

If the query passes this level of validation, the intermediate form of the query and all messages are attached to the QCB header to form an initial QCB. The QCB is passed to the QUERY-MONITORING-AND-CONTROL Subsystem, which will pass the QCB on to other nodes as required, and store it in the local dictionary.

3.2.3.2 Intrasubsystem Data Flows

Figure 3.2.3 shows the major data flows which occur within this Subsystem.

Subsystem 2: VALIDATE-TOKENIZED-QUERY

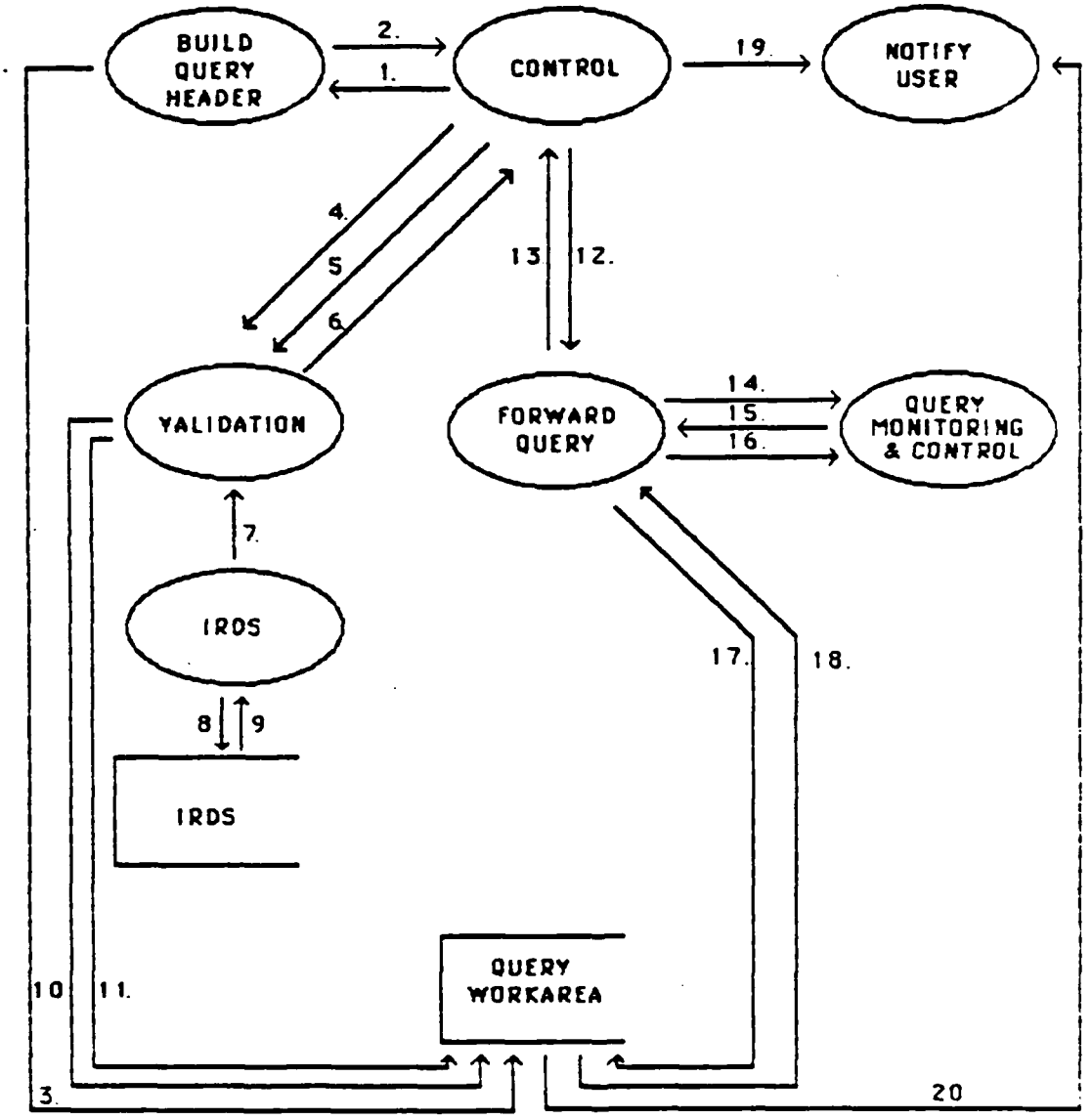


Figure 3.2.3

Subsystem 2: VALIDATE-TOKENIZED-QUERY

Data Flow	Data Item	Source Function	Destination Function
1.	Query-header	Control	Build Query Header
2.	Query-workarea-ptr	Build Query Header	Control
3.	Query-header	Build Query Header	Query Workarea
4.	Query-workarea-ptr	Control	Validation
5.	Tokenized-Query-body	Control	Validation
6.	Query-state	Validation	Control
7.	Local User Name Space (LUNS)	IRDS	Validation
8.	Dictionary Updates	IRDS	IRDS
9.	Dictionary Contents	IRDS	IRDS
10.	Intermediate-query	Validation	Query Workarea
11.	Diagnostics	Validation	Query Workarea
12.	Query-workarea-ptr	Control	Forward Query
13.	Forward-query-return-code	Forward Query	Control
14.	Destination-data-requests	Forward Query	QMC
15.	Destination-data	QMC	Forward Query
16.	Validate-intermediate-query-transaction	Forward Query	QMC
17.	Destination Data	Forward Query	Query Workarea
18.	Intermediate query	Query Workarea	Forward Query
19.	QCB-workarea-ptr	Control	Notify User
20.	Diagnostics	Query Workarea	Notify User

Figure 3.2.3

3.2.3.3 Tasking

Each input tokenized-query will result in the creation of a single task to validate it. Separate tasks exist for node input and output.

3.2.3.4 Exception Conditions

None.

3.2.3.5 Rationale for Critical Decisions

The following critical decisions were reached for this Subsystem:

1. Syntactic validation is performed at the user-node.

   There are three reasons for this:

   First, the output of syntactic validation is the tokenized-query, which is the standard format for different user interfaces to communicate to the IDN.

   Second, the tokenizing process refines communications costs, as the tokenized-query will be more condensed than a "raw" query passed to the IDN in a source format.

   Third, this approach is more appropriate for a "spreadsheet-like" user-interface. From the user's perspective, this type of interface is preferable to a "compiled" language, in that it provides for immediate correction of common errors. In such an interface, the user will interactively build the query. In this case syntactic errors can be easily identified while the user is building the query. Locating this validation at the user-node will also reduce transmission costs and processing overhead between the user-node and the IDN.

2. Complete semantic validation is not performed by this Subsystem.

   Complete semantic validation would require that a full representation of each Coherent Database Representation (CDR) be present at each D1 node. This is not desirable for several reasons:

   First, it would require replication of all CDRs at many more nodes than are required by the current proposal. As any given CDR changes, the changes would have to be propagated across many more nodes.

   Second, the metadata resident in the D1 dictionary is restricted to that which supports the user in using existing queries and defining new queries. The retention of additional metadata at a D1 node could compromise the "security perimeter" which the D1 nodes currently provide.

3. No user-verification is performed here.

   A user's entry to the IDN is checked at the user-node. Security markings for the user are passed along in a standard header. The IDN may append to this header, but may not change any values in it.

   This simplifies the approach to security, and provides additional protection against a D1 node administrator bypassing protections. The only security-related processing here is the binding of the Local User Name Space (LUNS) to the query.

4. Queries are defined in terms of "cells". (Note: The reader should review Section 3.2.4.1 in order to gain an understanding of the terminology used in the following paragraphs).

   The cell is a natural concept for a form, especially if a spreadsheet-like user interface is desired. Defining a form

in terms of nested cells also allows the display of data either horizontally or vertically, on a case by case basis. Presumably, data may be displayed horizontally in one cell, and vertically in another cell. Cells also provide a means for handling content-driven presentation variations.

5. Function arguments must be other form-elements, and those form elements must all be in either the same cell as the function or in a directly subordinate cell. If the function is specified in a cell which has no subordinates, there is an exception to this rule. In this case, all a function's arguments may also consist of LOCAL-ELEMENT-NAMES.

   The restriction on function arguments essentially says that one may do computations "horizontally" or "vertically". This is consistent with a spreadsheet-like interface. It also provides a very explicit mechanism for "rippling" calculations, which is natural for a form. Consequently, this restriction is not a significant one from the user's perspective. The exception permits the language to display summary-only data without any supporting detail. This is a common feature of many query languages, and as such, is also a reasonable feature for forms based query.

3.2.3.6 Open / Deferred Decisions

1. The native data types known to the query language must be specified. Tentatively the data types used by SQL have been assumed. It is possible that some other data types might be desired (such as DATE, TIME) and others (such as FLOAT) might not be desired. The architecture does not require that the same set of data types be used throughout the system. It is conceivable that some data types are used only by the local DBMS, and that a different set of data types are used by the query language and for data exchange.

2. The basic set of FUNCTIONs to be supported by the system has not been identified. Some functions which might be supported include:

```
CURRENT-DATE
CURRENT-TIME
MOD       (i.e., integer component of decimal or real)
REM       (i.e., decimal component of decimal or real)
MAX
MIN
COUNT
SUM       (i.e., sum of distinct data elements)
TOTAL     (i.e., of a given data element)
AVERAGE
MEAN
```

3.2.3.7 Critical Algorithms

(This discussion requires an understanding of the tokenized-query, which is discussed in detail in Section 3.2.4.1).

The following error conditions are to be identified:

- Duplicate form-elements error. A selection element may be displayed only once within all form-lines within a given cell. Computation elements must be unique within a form.

- Unknown references error. All labels identified in a selection criterion, computation expression, or function parameter list must be a LOCAL-ELEMENT-NAME known to the user, a FUNCTION name, or the name of a form-element.

o Inconsistent usage error.

- Functional notation (i.e., label, left parentheses, zero to n labels, right parentheses) may be used only if the label immediately to the left of the left parentheses is the name of a FUNCTION.

- LOCAL-ELEMENT-NAMEs may not appear to the left of the assignment-operator in a computational element.

- In a computational expression, the label to the left of the assignment operator must not be the name of a FUNCTION or a LOCAL-ELEMENT-NAME known to the user.

o Function argument usage error.

Given a computational-element E within a cell C, let F denote a FUNCTION used in the computational expression which defines E. Let a(1), ..., a(n) denote the arguments in the parameter list of F.

If C has subordinate cells, then:

- Each a(i) is a form-element.

- For each i, $1 \leq i \leq n$, a(i) is a form-element in C, or in C', where C' is a cell which is directly subordinate to C.

- If a(i) is a form-element in C, then each a(j), $j \neq i$, must also be a form-element in C.

- If a(i) is a form-element in C', directly subordinate to C, then a(j), $j \neq i$, must also be in C'.

If C has no subordinate cells, then:

- Each a(i) is either a form-element within C or a LOCAL-ELEMENT-NAME which is not specified in any cell C' superior to C.

- If a(i) is a form-element in C, then each a(j), $j \neq i$, must also be a form-element in C.

- If a(i) is not a form-element, then neither is a(j), $j \neq i$.

o Type conflict error.

For example, using arithmetic operators on form-elements whose types are not numeric is invalid. Exact specification of type conflict errors can only be done once the basic data-types used by the query-language is identified. See Section 3.2.3.6, point 3.

o Parameter error.

The proper number of parameters must be specified for each function.

3.2.4 DATA REQUIREMENTS

3.2.4.1 Explanation of Major Data Flows

Tokenized-query

This is the standard format for a query which is produced by the user interface. In this form, all symbols are tokenized;

extraneous white space has been removed, and the syntax states are known. Thus the query is syntactically correct upon entry to this Subsystem.

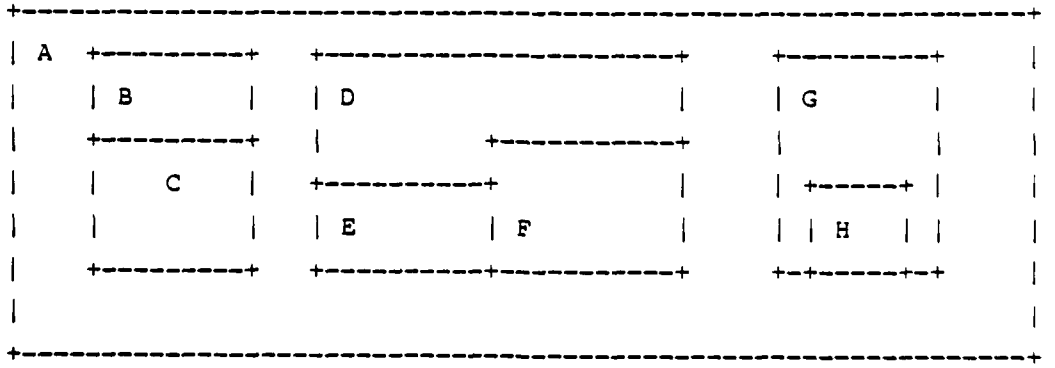
It is necessary to understand the basic concepts of query definition and organization before examining its data structure in detail. Accordingly, the following definitions are provided:

A **query** consists conceptually of a form definition and selection criteria.

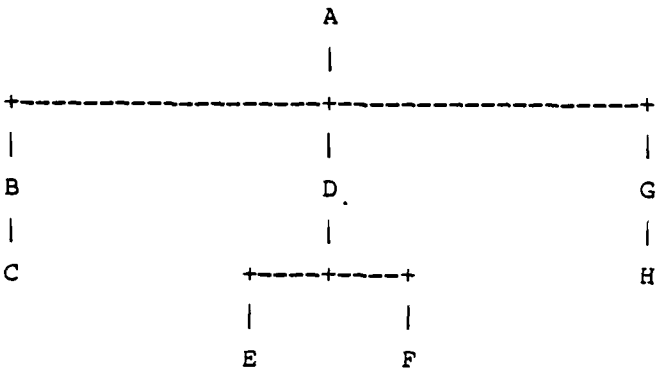
A **form** is a physical layout for the display of data. A form may be viewed as being divided into areas called **cells**.

A **cell** is a rectangular area within a form in which a logical grouping of data is displayed. Cell boundaries are defined by the coordinates of the top-left and bottom-right corners.

Cells may be *nested*. Informally this means that the boundaries of one cell lie entirely within the area of another cell. The boundary of an inner cell may coincide with 1, 2, or 3 boundaries of the outer cell. (If two cells have 4 boundaries which coincide, then there is no difference between them.) No cell boundary may cross the boundary of another cell. Cells may also abut (i.e., coincide at a single boundary without nesting.) Thus the following diagram show a valid placement of cells:



Cells then are ordered hierarchically. In the above example, the nesting hierarchy is as follows:



Note that in this case cells E and F abut.

If a cell X is nested within Y, it is said that is **subordinate** to Y, and it is denoted by X < Y. If there is some sequence of cells Z(1), ..., Z(n) such that X < Z(1) < ... < Z(n) < Y, it is said that X is **indirectly** nested within or subordinate to Y. When there are no intermediate cells it is said that X is **directly** subordinate to or nested within Y. Conversely, it is said that Y is **(directly or indirectly) superior** to X.

Cells contain one or more **form-lines.** A form-line is a set of one or more **form-elements**, which are labeled, one-dimensional character arrays which may contain alphabetic, numeric, or alphanumeric data.

Cells may **alternate.** (See Rationale statement: Section 3.2.3.5.) If cells C1 and C2 alternate, then in some cases cell C1 will be displayed, and in other cases, cell C2 will be displayed. Cells alternate based on the value of a form-element in a superior cell. The form-element which is used to determine which cell is displayed is called the **discriminant.** Thus alternating cells are analogous to variant records in PASCAL or Ada.

Cells have an **orientation**, which is either horizontal or vertical. In a horizontal cell, each form line is a row. In a vertical cell, each form-line is a column.

A cell has a **repetition factor.** The repetition factor identifies how many time each form line repeats within the cell. Thus it is not possible to have a single form-line, and a repeating form line within a single cell. In essence, this means that a cell contains a single kind of line which may wrap horizontally or vertically.

Form-elements may be classified as either **selection-elements** or **computation-elements.** A selection-element is identified by the local-user-name of a data element in the IDN database; it may or may not have any explicit selection criteria associated with it. A computation-element is identified by a label which is local to the query, and has the form label = expression. The expression may involve functions and other form-elements.

It is assumed that the user-interface enforces the constraints which are implicit or explicit within these definitions. (See Rationale statement: Section 3.2.3.5.)

Intermediate-query

The intermediate-query is the form of the query prior to its decomposition into canonical-subqueries. Should the query have to be restarted "from scratch", the intermediate-query will be the basis for this type of restart.

3.2.4.2 Special Data Requirements

None.

3.2.5 DATABASE REQUIREMENTS

3.2.5.1 Network Metadata

Each D1 processor must know:

Which D2Q Node is to service the Query.

Which other D1 Node(s) are to hold the QCB for restart.

3.2.5.2 Data Metadata

See Figure 3.2.4: D1 IRDS View.

3.2.5.3 Other

None.

3.2.6 NOTES

1. The QUERY-MONITORING-AND-CONTROL Subsystem has components at D1, D2Q, and D1* nodes. The D1 component of the QUERY-MONITORING-AND-CONTROL Subsystem performs the following functions in conjunction with this Subsystem:

   - For any query, it identifies the D2Q to perform the remaining validation and translation into canonical subqueries. Unless a node failure occurs, this becomes the controlling D2Q, noted as the D2Q/C.

D1 IRDS VIEW

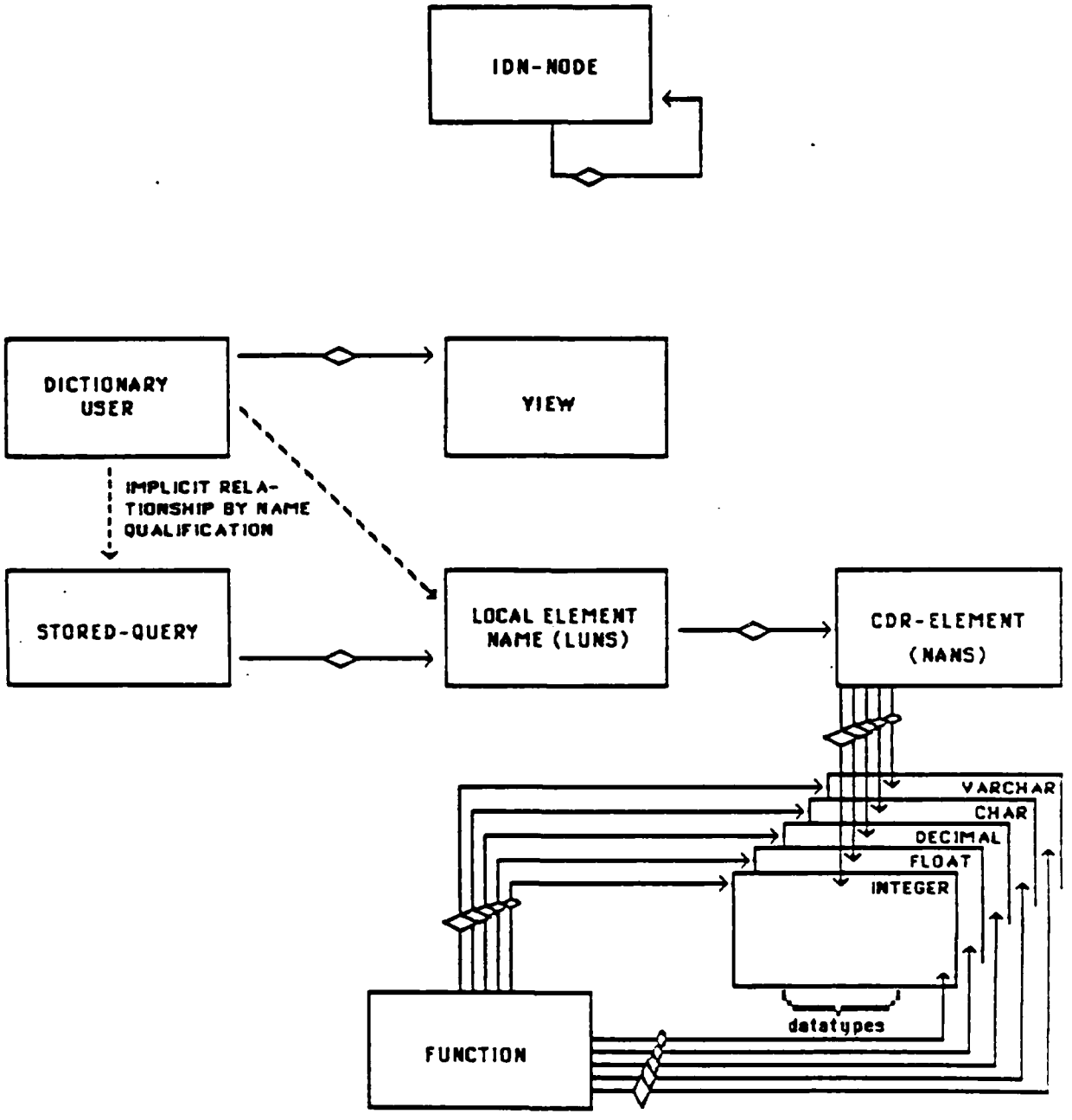


Figure 3.2.4

- It forwards the intermediate-query to the D2Q.

2. If, in the unlikely event that the targeted D2Q is unable to accept the intermediate-query the QUERY-MONITORING-AND-CONTROL Subsystem (QMC) sends the intermediate-query to an alternate D2Q.

(This page left intentionally blank)

3.3 SUBSYSTEM 3: VALIDATE-INTERMEDIATE-QUERY

3.3.1 OVERVIEW

Subsystem 3, the VALIDATE-INTERMEDIATE-QUERY Subsystem, accepts a intermediate-query which has been generated by the VALIDATE-TOKENIZED-QUERY Subsystem, and performs additional validation. The additional semantic validation includes:

- Conformance to value constraints.

  Value constraints consist of both range and value-list criteria.

- Conformance to location constraints.

  A location constraint is a requirement that one or more CDR-ELEMENTs must reside at a particular node. It is specified by a virtual-key expression in selection criteria.

- Conformance to data-grouping constraints.

  A data grouping constraint is a requirement that two or more CDR-ELEMENTs must exist in the same CDR or CDR-RECORD-TYPE.

In the process of performing this semantic validation, the QCB components built by this Subsystem include:

- An incomplete Subquery Control Block (SQCB) for the final assembly of the query output.

- Partially completed Input File Descriptions for each CDR to be accessed in the query.
- A Relation Description for each Input File Description.

3.3.2 INTERFACE SPECIFICATIONS

3.3.2.1 Intersubsystem Data Flow Diagram

Figure 3.3.1 shows the data flows between this and other Subsystems.

3.3.2.2 Subsystem Events

Figure 3.3.2 shows the major events which occur in this Subsystem.

3.3.3 TECHNICAL REQUIREMENTS

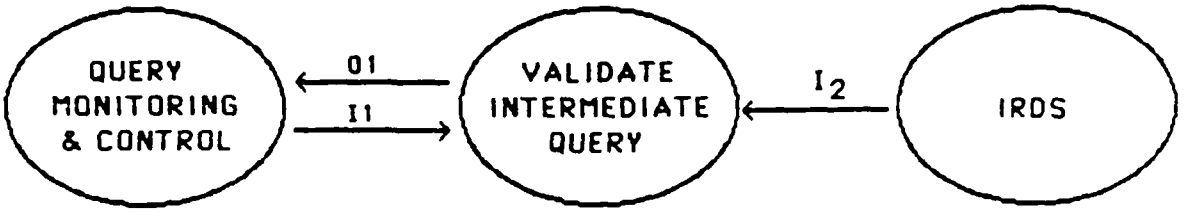
3.3.3.1 Actions

Validation of the intermediate query proceeds according to the rules identified in Section 3.3.3.7. Any exception conditions are noted and corresponding diagnostic messages are placed in the Query Control Block (QCB).

3.3.3.2 Intrasubsystem Data Flows

Figure 3.3.3 shows the major data flows which exist within this Subsystem.

SUBSYSTEM 3: VALIDATE-INTERMEDIATE-QUERY

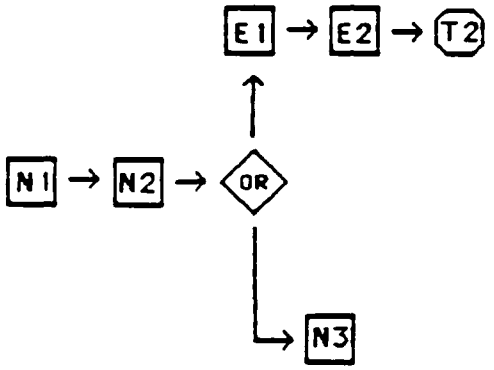


In	Data Item	Source
I1	Validate-intermedate-query-transaction	Query Monitoring & Control
I2	CDRs	IRDS

Out	Data Item	Destination
O1	QCB	Query Monitoring & Control

Figure 3.3.1

Subsystem 3: VALIDATE-INTERMEDIATE-QUERY



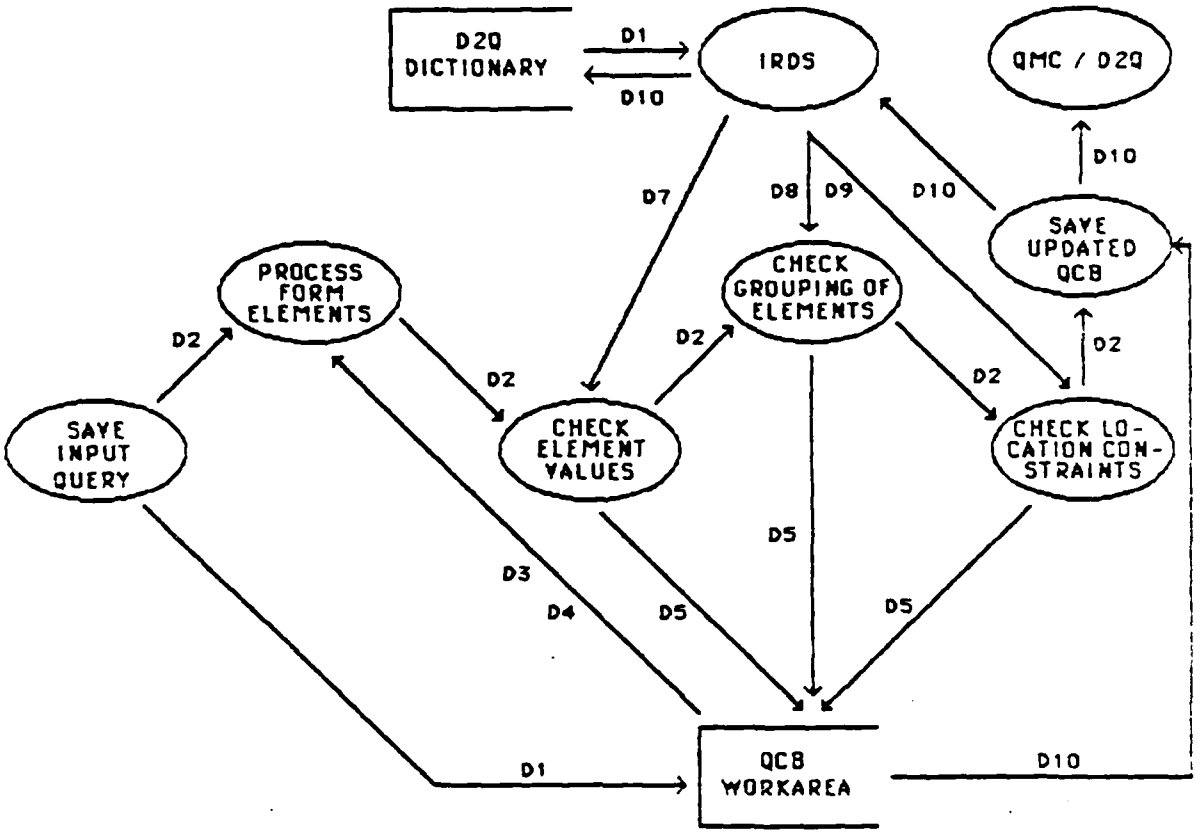
Normal	Event Description
N1	Task initiated for normalized query validation
N2	Normalized-query is validated
N3	Subsystems 4-5 execute

Error	Event Description
E1	Diagnostic Message Output
E2	QCB is purged locally

Terminal	Event Description
T1	Subsystem 3 completes processing

Figure 3.3.2

SUBSYSTEM 3: VALIDATE-INTERMEDIATE-QUERY



Data Flow	Data Item	Source Function/Store	Destination Function/Store
D1	Intermediate-query	Save Input Query	QCB Workarea
D2	QCB Workarea PTR	Save Input Query	Process Form Elements
	QCB Workarea PTR	Process Form Elements	Check Element Values
	QCB Workarea PTR	Check Element Values	Check Grouping of Elements
	QCB Workarea PTR	Check Groupings or Elements	Check Location Constraints
	QCB Workarea PTR	Check Location Constraints	Save Updated QCB
D3	Query Symbol Table	QCB Workarea	Process Form Elements
D4	Selection Criteria	QCB Workarea	QCB Workarea
D5	Diagnostic/Error Messages	Check Element Values	QCB Workarea
	Diagnostic/Error Messages	Check Grouping of Elements	QCB Workarea
	Diagnostic/Error Messages	Check Location Constraints	QCB Workarea
D6	D1Q Dictionary Descriptors	D2Q Dictionary	IRDS
D7	CDR Elements and Value-Sets	IRDS	Check Element Values
D8	Functional Dependency Matrices	IRDS	Check Grouping of Elements
D9	CDR-element-to-CDR-relationship	IRDS	Check Location Constraints
D10	Updated QCB	QCB Workarea	Save Updated QCB
	Updated QCB	Save Updated QCB	IRDS
	Updated QCB	Save Updated QCB	QMC/D2Q
	Updated QCB	IRDS	D2Q Dictionary

Figure 3.3.3

3.3.3.3 Tasking

One task is created for each intermediate query which is processed.

3.3.3.4 Exception Conditions

None.

3.3.3.5 Rationale for Critical Decisions

1. Data value and location criteria are checked at the D2Q rather than the D1 processor.

   The metadata which describes where CDR-ELEMENTs reside, and any partitioning of their values according to Delegated Production Policy (DPP) is stored at D2Q nodes rather than at D1 nodes. Therefore, this validation must be done at the D2Q.

   Placing this metadata at D2Q nodes prevents a user from determining the residence of any data item, since the user has no facilities to access a dictionary at a D2Q, except for executing a query. This architecture thus provides an effective security perimeter which prevents users from deliberately or inadvertently obtaining knowledge about the residence of data elements.

2. The QCB for the query is shipped to each shadow D2Q (D2Q/S) after passing all validation (Subsystems 3 through 5).

   The D2Q Node which controls the processing of the query is called the "controlling D2Q" and is denoted by D2Q/C. Subsystems 3 through 6 all execute at the D2Q/C. Shadow D2Qs are those which are assigned to act as a backup for a given D2Q.

If the QCB were shipped to each D2Q/s prior to completing validation, then either the query would have to be validated at each D2Q; or the QCB's at each D2Q/S would have to be replaced or purged after the completing Subsystems requires that the query be resubmitted. This is acceptable because the cost of resubmission is for less than the overhead incurred in trying to provide restart prior to completing Subsystem.

3.3.3.6 Open / Deferred Decisions

1. The interfacing of Subsystems 3 through 5 will be optimized in detail design.

   The present specification assumes for simplicity that each of these Subsystems is executed sequentially under the control of the D2Q component of the QUERY-MONITORING-AND-CONTROL Subsystem. This may change in detail design. For example, it may be more desirable to have a single task devoted to a query which combines Subsystems 3 through 5.

2. "Compiled" Meta data at the D2Q.

   In Section 3.3.4.2 reference is made to functional-dependency matrices as at least part of the output of "compiling" CDRs. No decision has been made as to whether or not the matrices should be the only information compiled at the D2Q.

3. Versioning requirements.

   The requirements for "versioning" CDRs or configurations of CDRs must be clarified. A basic assumption regarding the key values of target databases in the IDN which has been made in order to greatly simplify the processes of Query Decomposition and Network Data Request Language generation and execution is:

> If A is key in any database; i.e.; is used for unique identification within that database, then A may be used as a key in any database in which it appears.

This assumption is valid as long as a data element type which occurs in multiple databases can be used as a primary key in one database and can serve as a "foreign key" in the other records in which it appears. If the data element type does not uniquely define a record in some database, the Coherent Database Representation for the database may indicate that another element value appearing in the query may be used as a unique key or to construct a unique concatenated key. Otherwise, a non-unique key may have to be used. This may result in the production of multiple output records in response to the query.

In order to ensure the unique identification of the source of the records generated in response to a subquery, "virtual keys" will be associated with every database key defined in the CDRs. This concept can be exemplified by considering how telephone directories are referenced. Each directory contains a list of unique telephone numbers for the given area. These numbers are only unique, however, because of an implied area code (i.e., "virtual key").

Likewise, each file, table, and record set in each database will have an associated "virtual key" which will be concatenated with the record(s) extracted in response to a subquery. With this unique internal identification, the IDN query processing system will be able to correctly control the merging of sub-responses into a single, uniform, composite response. The introduction of "virtual keys" is necessitated by the decentralized, rather than distributed, nature of the databases in the IDN. They permit discrimination between responses from different databases and permit the semantics of these responses to be reconstructed for user output composition.

Accordingly, given this assumption on database keys, changing the keys in one CDR may require changing keys in other CDRs.

3.3.3.7 Critical Algorithms

In order to express the rules clearly and succinctly, the following notation will be used:

Let Q denote the query. Let C(i) denote cells in Q. If C(i) is subordinate to C(j), denote this by C(i) < C(j).

Let R be a relation composed of selection-elements. Let K -> R denote that K is a key for a relation R. This is equivalent to saying that for each element r of R, there exist "key elements" k(1), ..., k(n) such that each k(i) is in R and K, and that r is functionally dependent on k(1) through k(n).

Validation criteria for grouping can now be specified: (See Section 3.3.6: Notes 1 and 3.)

1. Let C(1), ..., C(n) be a nesting of cells such that C(n) < C(n-1) < ... < C(i+1) < C(i) < ... < C(2) < C(1). For each C(i) let R(i) be the relation composed of selection-elements within each C(i). Let R be the relation R(1) x R(2) x ... x R(i) x ... x R(n). Then there exists a relation K within R such that K -> R.

2. For every C(i) a cell in Q, let s(i,j) denote the selection-elements of C(i). Let D(1), ..., D(m) denote CDRs in which s(i,j) participate. Then for each D(x), there exists a relation K'(x), which functionally determines s(i,j), and each k in K'(x) is in D(x) and is a selection-element in C(i) or some cell superior to C(i).

3. Let C(i), s(i,j), D(1), ..., D(m), K'(x) be as in rule 2 above. Then each k in K'(x) is in each D(y), y ≠ x, and K'(x) functionally determines s(i,j) in D(y). (See notes 2 and 3).

3.3.4 DATA REQUIREMENTS

3.3.4.1 Explanation of Major Data Flows

Query-control-block (QCB)

The QCB goes through several states during the execution of this and later Subsystems. The form of the complete QCB is defined with the data-flow definitions. Figure 3.3.4 shows the major sections of the QCB, and how they relate to each other.

The **QCB** is the aggregate of all the components identified below. The **intermediate-query** is stored as part of the QCB to facilitate restart if required. Administrative and Diagnostic Messages are stored with the QCB in a Message Queue. This acts as a message queue. This facilitates performing more complete validation, and provides a history of the query's decomposition and execution.

The **Subquery Control Block (SQCB)** is the aggregation of the **Subquery Control Block Header**, and all related **Input File Descriptions**, **Output File Descriptions**, and **Relation Descriptions**. The **Execution Graph** is the collection of all Subquery Control Blocks.

There is a partial ordering of SQCBs in the Execution Graph by Input File Descriptions. In this partial ordering, a given SQCB (A) is "after" any other SQCB (B) whose Output File Description is associated with an Input File Description of A. In this context, a "file" is a subquery's access-path to data. With the exception of the final assembly's output file, each file has only one corresponding Relation Description.

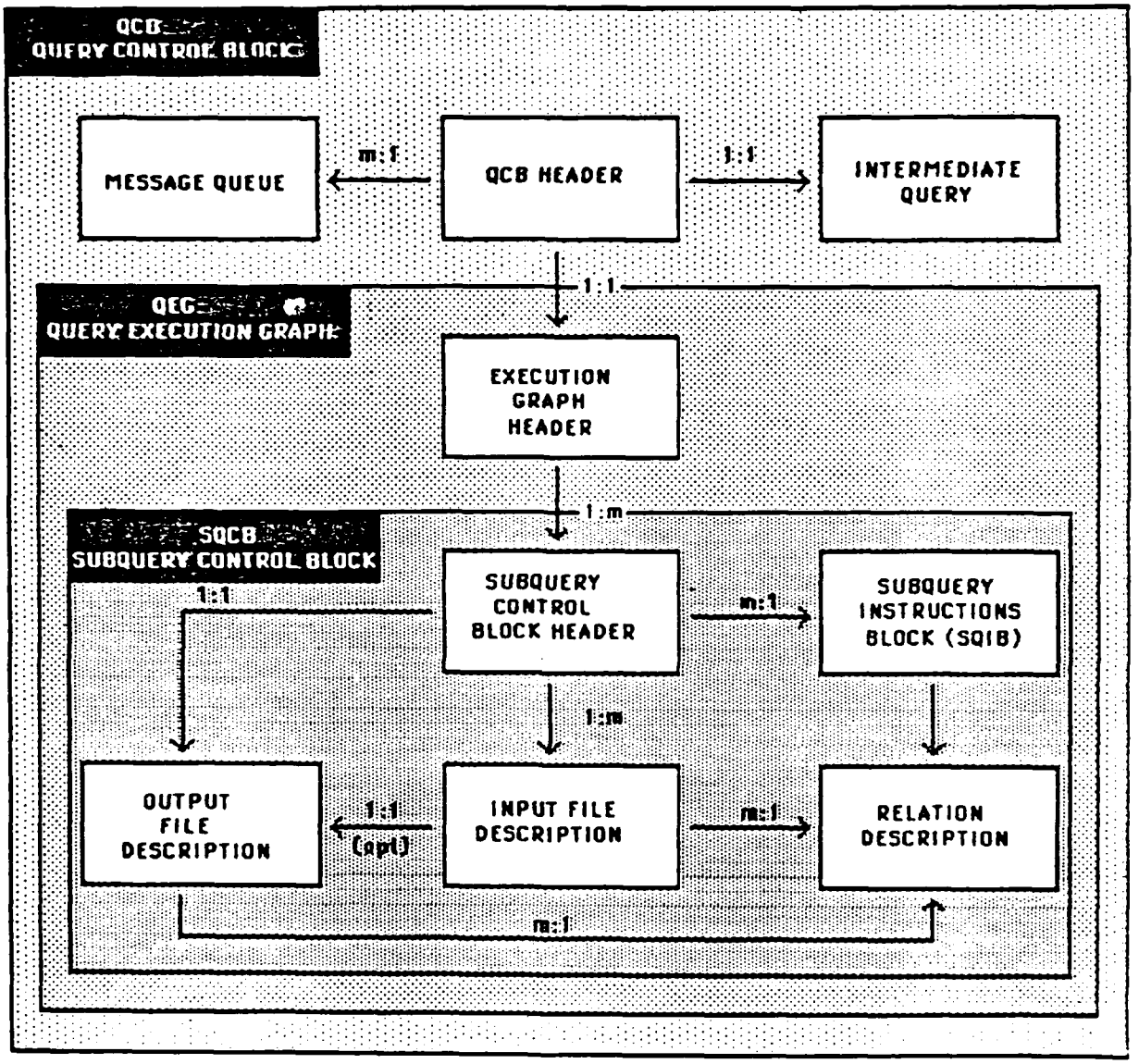

QCB
QUERY CONTROL BLOCK
MESSAGE QUEUE
m:1
QCB HEADER
1:1
INTERMEDIATE QUERY
1:1
QEG
QUERY EXECUTION GRAPH
EXECUTION GRAPH HEADER
1:m
SQCB
SUBQUERY CONTROL BLOCK
1:1
SUBQUERY CONTROL BLOCK HEADER
m:1
SUBQUERY INSTRUCTIONS BLOCK (SQIB)
1:m
OUTPUT FILE DESCRIPTION
1:1
(opt)
INPUT FILE DESCRIPTION
m:1
RELATION DESCRIPTION
m:1

Figure 3.3.4

The QCB goes through various stages of completion as the query is validated and executed. The major stages of QCB completion are as follows:

- The initial QCB

  The initial QCB consists only of the QCB header and the intermediate query.

- The QCB resulting from Subsystems 3 through 5.

  When the structure of the query is validated against the structure of the various CDRs corresponding to IDN databases, a relation description is constructed for each query cell. As the query is decomposed into canonical subqueries, a Subquery Control Block Header is created. Input and output Relation Descriptions are connected to the corresponding canonical subquery instructions. When a CDR resides at multiple nodes, multiple Subquery Control Block Headers are built, and each one references the same set of relation descriptions and subquery instructions. Different subqueries, however, will have different Input and Output File Descriptions. Each subquery may have multiple Input File Descriptions but will have only one Output File Description. Normally, File Descriptions will correspond to only one Relation Description. The single exception to this is the output file for final assembly, which will have one Relation Description per cell.

  The completed QCB will be in either a valid or error state. If it is in a valid state, then it will contain a completed execution graph. This QCB is stored at the primary D2Q Node and all its respective shadow D2Q Nodes.

  If it is in an error state, then all error messages and all the SQCBs which were produced will be stored in the QCB until errors prevented completion of the execution graph. This QCB

is retained only at the primary D2Q Node, and only until all error messages have been accepted by the user node which initiated the query.

- The QCB throughout Subsystems 7 through 14.

As subqueries execute, status information on each subquery is maintained in each completed QCB at each D2Q Node which contains the QCB for the query. This information includes the identity of the D1* assigned to process the subquery, whether the subquery is executing, waiting, or has completed execution. Upon completion, the name and amount of data of each output file is stored in the Output File Description. When a file is input to another subquery, the corresponding Input File Description is updated with the name of the physical file at the receiving node, (i.e., where the next subquery is to execute.) Note that since several subqueries may execute at the same node, the receiving node may be the same as the sending node.

3.3.4.2 Special Data Requirements

Functional Dependency Matrices

Checking data grouping constraints could be very time consuming if validation involved reading relationships involving CDRs, CDR-RECORD-TYPEs, CDR-KEYs, and CDR-ELEMENTs. Accordingly, it is desirable to "compile" functional dependencies into a more compact form. This compiled form could become memory-resident for fast processing of the data grouping constraints.

Given n CDR-ELEMENTs and CDR-KEYs, and m CDRs, then m, bit-mapped n by n matrices can represent functional dependencies within the CDRs. Using the same size of matrix for each CDR enables each CDR-ELEMENT to be identified by the same position in each CDR.

3.3.5 DATABASE REQUIREMENTS

3.3.5.1 Network Metadata

None.

3.3.5.2 Data Metadata

The diagram on the D2Q IRDS View, shown in Figure 3.3.5, provides an overview of the contents of the D2Q dictionary.

Usage of Entity and Relationship-types by Type of Validation

The following entity-types are used by this Subsystem for validating the query's conformance to data grouping constraints:

CDR
CDR-RECORD-TYPE
CDR-KEY
CDR-ELEMENT

The following relationship-types are used by this Subsystem for validating the query's conformance to data grouping constraints:

CDR-SET-OF-CDR-RECORD-TYPE
CDR-RECORD-TYPE-HAS-CDR-KEY
CDR-KEY-SET-OF-CDR-ELEMENT
CDR-RECORD-TYPE-SET-OF-CDR-ELEMENT

(These entities and relationships would be used to construct functional dependency matrices - see Section 3.3.4.2.)

D2Q IRDS VIEW

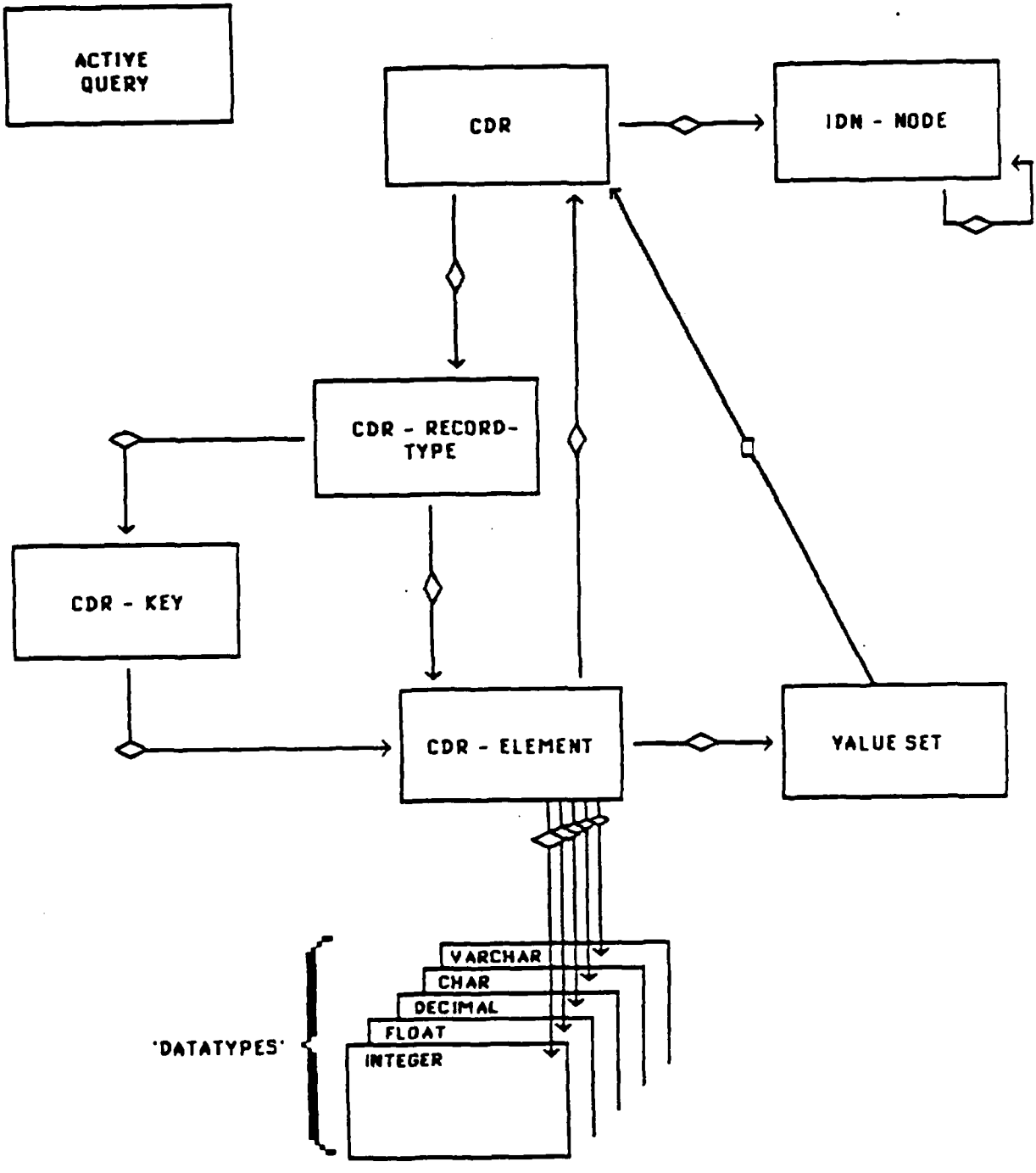


Figure 3.3.5

Value constraint validation requires the following types of entities and relationships:

CDR-ELEMENT
VALUE-SET
CDR-ELEMENT-HAS-VALUE-SET

Validating a query's conformance to location constraints uses the following relationship-types in addition to all those previously mentioned:

VALUE-SET-IN-CDR
CDR-ELEMENT-IDENTIFIES-CDR (See Virtual Key Elements)

VALUE-SET Entity-type

Since value-lists and ranges of values are part of the D2Q dictionary, but not the D1 dictionary, it is necessary to "decouple" this meta-data from the CDR-ELEMENT. Accordingly, the VALUE-SET entity-type has been created.

Since certain values for a CDR-ELEMENT may be restricted to different CDRs, a relationship-type relating CDR to VALUE-SET is required. Since different CDRs may contain different sets of values for a given CDR-ELEMENT, a given CDR-ELEMENT may be related to multiple VALUE-SET entities. Delegated Production Policy (DPP) dictates that for a given subrange of values, a user query will access a given CDR. This implies that disjoint subranges should be defined by distinct VALUE-SET entities, and each VALUE-SET entity should be connected to at most one CDR (excluding the possibility of multiple versions).

Virtual Key Elements

It is possible that different users could have knowledge of different virtual keys. This could be handled by:

- o A separate VIRTUAL-KEY entity-type.
- o Using a special set of CDR-ELEMENTs to define virtual-keys.

In either case, each user's LUNS would contain a LOCAL-ELEMENT-NAME which would be related to the entity which defines the set of virtual-keys needed. This entity would then have to be related to each such CDR, (which is where the virtual-key value is uniquely defined).

In analyzing the two alternatives, the second is preferable as it does not impact the D1 dictionary and Subsystem 2. Thus the following relationship-type is added to the schema:

CDR-ELEMENT-IDENTIFIES-CDR.

3.3.5.3 Other

None.

3.3.6 NOTES

1. For clarification, the data grouping validation criteria may be rephrased as follows:

   The first validation criterion basically says that whenever a cell is nested within another, a one-to-many relationship exists between the superior cell's contents and the subordinate cell's contents.

The second validation criterion says that for any given selection-element, a key can be constructed from other selection-elements in the same or superior cells will enable access to the given selection-element in each CDR in which it appears.

The third validation criterion is even stronger than the second. It says that if an element is in two or more CDRs, it can be identified by the same keys in each CDR. This is a reformulation of the assumptions regarding database keys identified earlier in this Chapter. It is possible that the first and second criteria are alone sufficient to provide reasonable query responses. Even if this is the case, however, a "compiled" form of the CDRs might be desirable.

Note that these criteria do not imply that all non-key selection-elements within a given cell reside in the same CDRs.

2. The assumption about database keys (see Note 2 above and the third data grouping validation criterion) may be better validated as part of "compiling" the CDRs into the functional dependency matrices identified in the Special Data Requirements section of this chapter. This would suggests that only one functional dependency matrix be compiled for all CDRs at the D2Q.

3. The criteria regarding selection-elements in computation-expressions and FUNCTION parameters specified in Subsystem 2 are sufficient to allow restricting data grouping validation criteria only to selection elements.

3.4 SUBSYSTEM 4: DETERMINE-DATA-ACCESSIBILITY

3.4.1 OVERVIEW

Subsystem 4, the DETERMINE-DATA-ACCESSIBILITY Subsystem, determines which data nodes are to be accessed. In the process of determining this, five tables are added to the QCB. These tables are used in Subsystem 5, the DECOMPOSE-QUERY-INTO-CANONICAL SUBQUERIES Subsystem.

3.4.2 INTERFACE SPECIFICATIONS

3.4.2.1 Intersubsystem Data Flow Diagram

Figure 3.4.1 shows the data flows which exist between this and other Subsystems.

3.4.2.2 Subsystem Events

Figure 3.4.2 shows the major events which occur within this Subsystem.

3.4.3 TECHNICAL REQUIREMENTS

3.4.3.1 Actions

The processing of this Subsystem proceeds in four phases:

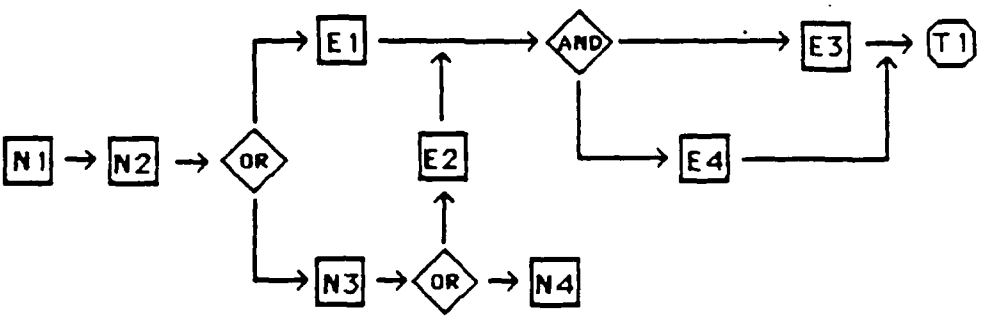
Subsystem 4: DETERMINE-DATA-ACCESSIBILITY

In #	Data Item	Source
Input 1	QCB	Query Monitoring & Control
Input 2	CDR	IRDS
Input 3	Static-network-view	IRDS
Input 4	Dynamic-network-view	Query Monitoring & Control

Out #	Data Item	Destination
Output 1	QCB	Query Monitoring & Control

Figure 3.4.1

Subsystem 4: DETERMINE-DATA-ACCESSIBILITY



Normal	Event Description
N1	Lists built
N2	Extraneous list elements purged
N3	Selection criteria scanned for references to inaccessible values
N4	IDN nodes assigned to each CDR used in query

Error	Event Description
E1	Error detected: LUNs and NANs not in sync
E2	Security violation: References made to Inaccessible Values
E3	All messages output to appropriate destination
E4	QCB purged locally

Terminal	Event Description
T1	Query terminates

Figure 3.4.2

I Accessing the dictionary to build lists in the QCB. The lists identify all CDR-ELEMENTS, CDR-RECORD-TYPES, and CDR's which are either explicitly or implicitly referenced in the query.

II Purging from the lists all those entities which are not accessible because of security constraints.

III Checking the selection-criteria against the reduced lists to determine if any secured data was requested.

IV Determining which IDN-nodes should be accessed.

Phase I

In the first phase, the query symbol table is scanned for all selection-elements. The selection-elements are processed as follows:

- For each selection element, the D2Q dictionary is accessed to find all CDR-RECORD-TYPES which contain the specified CDR-ELEMENT.

- For each CDR-RECORD-TYPE, all CDRs which contain the specified CDR-RECORD-TYPE are identified.

As the dictionary is accessed, five tables are built in the QCB:

1. The E-LIST, which has an entry for each selection-element.

2. The ER-LIST, which has one entry for each CDR-RECORD-TYPE to CDR-ELEMENT relationship accessed.

3. The R-LIST, which has one entry for each CDR-RECORD-TYPE accessed.

4. The RC-LIST, which has one entry for each CDR to CDR-RECORD-TYPE relationship accessed.

5. The C-LIST, which has one entry for each CDR accessed.

The E-LIST and R-LIST also contain usage counts which identify respectively how many CDR-RECORD-TYPEs contain each CDR-ELEMENT, and how many CDRs contain each CDR-RECORD-TYPE.

Phase II

In the second phase of this Subsystem, each the query's security-markings is checked against those of each CDR in the C-LIST. If the query's security markings do not authorize access to the CDR, then:

- The CDR is removed from the C-LIST.
- For each removed CDR, the corresponding RC-LIST entries are also removed. For each removed RC-LIST entry, the counter in the corresponding R-LIST entry is decremented.
- Within the R-LIST, any entry with a zero usage counter is removed.
- Then within the ER-LIST, any entry which references a CDR-RECORD-TYPE which is no longer represented in the R-LIST is removed. For each ER-LIST entry removed, the usage counter in the corresponding E-LIST entry is decremented.

If any entry in the E-LIST has a zero usage counter, the integrity of the Local User Name Space (LUNS) has been lost. (See Exception Condition 1 in Section 3.4.3.4) In this event, appropriate messages are produced and stored in the QCB for all such CDR-ELEMENTS so identified in the E-LIST, and query processing terminates.

Phase III

Phase III is entered only if all entries in the E-LIST have non-zero usage counts. In this Phase, selection criteria are scanned for references to inaccessible values or ranges of values. For each selection-element which specifies particular values, the specified values are matched against the corresponding VALUE-SET entities. For each VALUE-SET entity which covers a directly referenced value, the corresponding CDR is obtained (if any). If any such CDR is not specified within the C-LIST, a security violation exists. In this case, a message indicating that the requested data is not currently available is provided for the user. Other messages which indicate security violation are produced for security administrators. All messages are stored in the QCB. Algorithm 1 in Section 3.4.3.7 describes security validation processing in detail.

Phase IV

After completion of the third phase, the fourth phase associates an IDN-node to be accessed with each CDR remaining in the C-List.

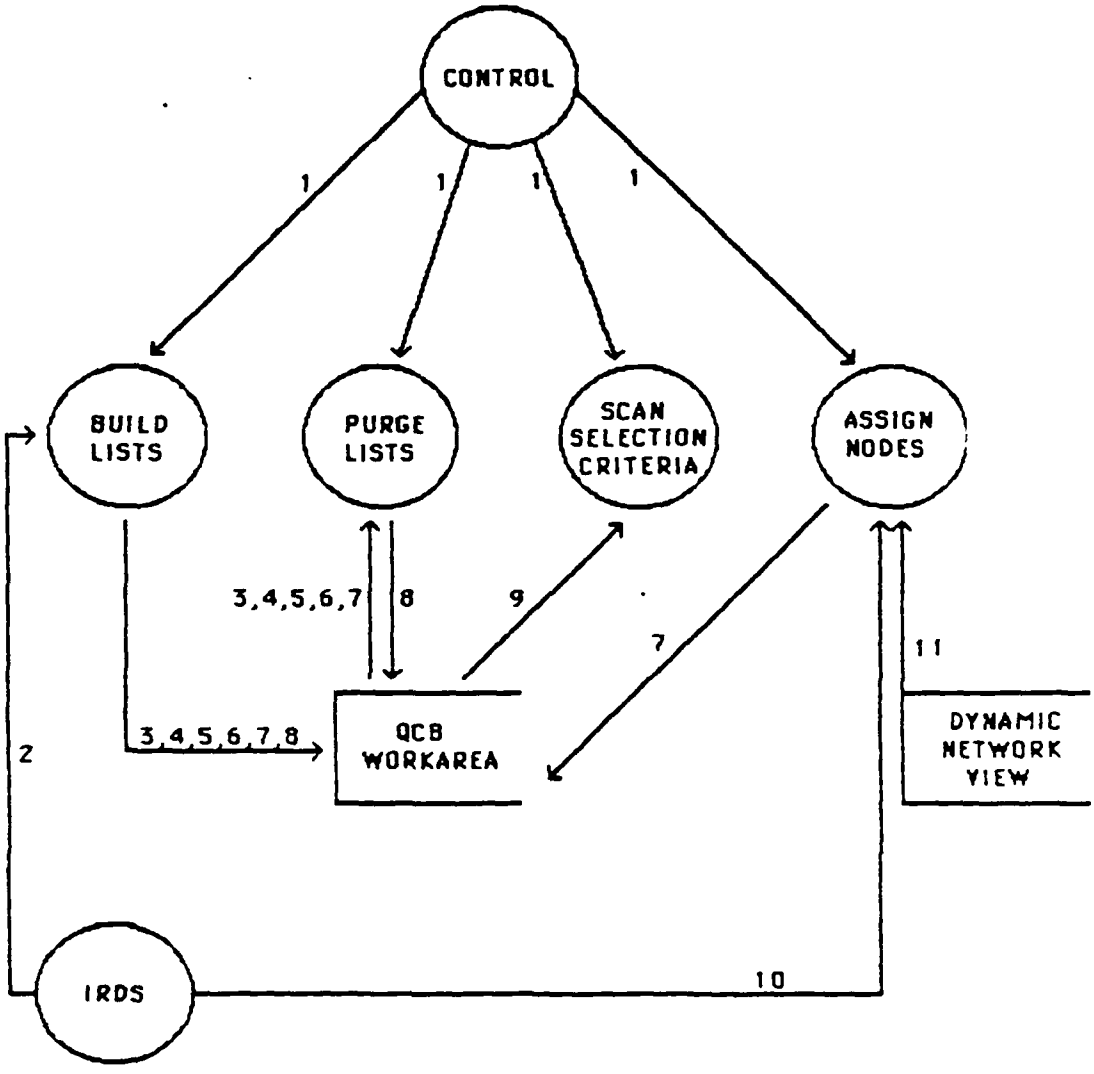
3.4.3.2 Intrasubsystem Data Flows

Figure 3.4.3 shows the major processing components of this Subsystem and the data flows between those components.

3.4.3.3 Tasking

A single task exists for each query.

SUBSYSTEM 4: Determine Data Accessibility



1 QCB Address
2 CDR entities and relationships
3 E_List
4 ER_List
5 R_List
6 RC_List
7 C_List
8 Diagnostics
9 Intermediate query
10 Static network view
11 Dynamic network view

Figure 3.4.3

3.4.3.4 Exception Conditions

1. If any entry in the E-LIST has a zero usage counter, the user's Local User Name Space (LUNS) contains a LOCAL-ELEMENT-NAME which is associated with a CDR-ELEMENT which is inaccessible according to security constraints on the target database. Thus the integrity of the LUNS has been lost. As such, some major corrective action is required. This corrective action would probably be to remove the offending LOCAL-ELEMENT-NAME(s) from the Local User Name Space. In this event, appropriate messages are produced and stored in the QCB for all such CDR-ELEMENTS so identified in the E-LIST. In addition to error messages for the user, messages are generated for .dministrators with responsibility for security, D2Q nodes, and the effected D1 nodes.

   Since the above situation results from a problem in administration, the user will receive a message indicating that the desired information is currently not available.

3.4.3.5 Rationale for Critical Decisions

1. The reason for terminating the query in the case of Exception Condition 1 is that there has been some error in the content of the dictionaries of the IDN. The severity of the error demands immediate attention. Repeated failure of the query will motivate the user to contact the D1 node administrator.

2. Algorithm 1, identifying security violations, is motivated by the following considerations:

   Suppose a CDR-ELEMENT has three VALUE-SET entities which cover the ranges (1,100), (101,200), (201,300). In this case, each VALUE-SET entity has a corresponding CDR. Now suppose the user's security markings allow access to the first and third CDR.

The user can quite innocently select data based on the range (50, 250).

On the other hand, testing equality or inequality of specific values, or a subrange of values which is associated with an inaccessible CDR indicates that the user either knows of certain values which he should not have knowledge of, or that the user is probing the system.

3. Algorithm 1 assumes that it is valid for a user to request information from a CDR other than the one defined by DPP by using a virtual-key-restriction-clause. The rationale is that the user might be interested in alternate sources.

3.4.3.6 Open / Deferred Decisions

1. Phases I and II can proceed in either breadth-first or depth-first fashion. In the breadth first-approach, phase one would fill in the E-LIST and ER-LIST; proceed to the R-LIST and RC-LIST, and finish with the C-LIST. In the depth-first approach, entries would be constructed in each table for each CDR-ELEMENT. The decision for selecting one approach over another should be based on the actual implementation of the various lists and the physical database design of the dictionary itself.

   Depending on the number of CDRs, the following approach may be more efficient. First scan all CDRs, placing only those which match the user's security markings in the C-LIST, then expanding the CDRs to fill in the RC-LIST and R-LIST. Then fill in the E-LIST and ER-LIST, and purge unnecessary entries of the R-LIST.

2. The choice of how to implement the various lists is deferred to detail design. While linked-lists are a possible implementation, other implementation strategies might be desirable. The approach

to implementing these is dependent on the storage medium for the QCB. For the present, the lists may be thought of as simple tables.

3.4.3.7 Critical Algorithms

1. The following algorithm is used to check selection criteria for security violations:

SELECTION-CRITERIA-SECURITY-CHECK PROCEDURE:

```
Select a selection-element until done.

    For each selection-element:

        Use symbol-table to find selection-clause.

        If there is no selection clause:

            Proceed to next symbol-table reference or
            next selection-element.

        Else:

            If there is a virtual-key restriction-clause:

                Determine if the virtual-key identifies
                a CDR in the C-LIST.

                If the corresponding CDR is in the C-LIST:

                    Proceed.
```

Else:

Add security-violation messages to QCB.

Do VALUES-SECURITY-CHECK.

End procedure.

VALUES-SECURITY-CHECK PROCEDURE:

Parse selection-clause.

If a relational-expression expresses a range:

Find all VALUE-SET entities which specify values or subranges which intersect the range.

If any of the VALUE-SET entities are associated with an inaccessible CDR:

If the boundary values of the range are associated with accessible CDRs,
or
if the range-criteria is "ANDed" with a virtual-key restriction-clause:

Proceed.

Else:

Add security-violation error messages to the QCB.

If a relational-expression tests only for equality or inequality or values are specified in set-membership test
and
the expression is not "ANDed" with a virtual-key restriction clause
and
the value is within a subrange of values associated with an inaccessible CDR:

Add security-violation error messages to the QCB.

End procedure.

3.4.4 DATA REQUIREMENTS

3.4.4.1 Explanation of Major Data Flows

QCB

This has been defined previously, in 3.2 Subsystem 2, VALIDATE-TOKENIZED-QUERY.

E-LIST

The E-LIST contains an identifier for the CDR-ELEMENT, and a counter. The counter identifies the number of CDR-RECORD-TYPES to be accessed which contain the CDR-ELEMENT.

ER-LIST

The ER-LIST is a cross-reference list, which identifies CDR-ELEMENT to CDR-RECORD-TYPE relationships. It associates the entries in the E-LIST with entries in the R-LIST.

R-LIST

The R-LIST is a table which identifies each CDR-RECORD-TYPE which is to be accessed in the query. The R-LIST contains a counter which indicates how many CDRs contain the specified CDR-RECORD-TYPE.

RC-LIST

The RC-LIST is another cross-reference list. It associates entries in the R-LIST with entries in the C-LIST.

C-LIST

The C-LIST identifies each CDR to be accessed by the query. In addition to the CDR's name, it also contains the CDR's security markings.

3.4.4.2 Special Data Requirements

None.

3.4.5 DATABASE REQUIREMENTS

3.4.5.1 Network Metadata

The dynamic network view is used to determine the IDN node to be assigned to the CDR.

3.4.5.2 Data Metadata

See D2Q IRDS View in 3.3 Subsystem 3: The VALIDATE-INTERMEDIATE-QUERY.

3.4.5.3 Other

None.

3.4.6 NOTES

1. There is a need to prevent the situation identified in Exception Condition 1 (Section 3.4.3.4) from occurring. The difficulty lies in the fact that users' security markings are defined in the D1 dictionary, and the security markings for CDR are defined in the D2Q dictionary. When a D1 node administrator establishes or maintains a local user name space, it is necessary for the administrator to know which CDR-ELEMENTs may be assigned to the local user name space. The approach for dealing with Exception Condition 1 will be addressed in the IDNSF (Integrated Data Network Support Facility).

3.5 *SUBSYSTEM 5: DECOMPOSE-QUERY-INTO-CANONICAL-SUBQUERIES*

3.5.1 OVERVIEW

Subsystem 5, the DECOMPOSE-QUERY-INTO-CANONICAL-SUBQUERIES Subsystem, uses the information obtained in the previous Subsystem (the DETERMINE-DATA-ACCESSIBILITY Subsystem) to decompose the query into one or more canonical-subqueries. In this Subsystem:

- The CDR-RECORD-TYPEs which must be accessed for each cell are identified.
- Subquery Instruction Blocks (SQIBs) are generated for each CDR-RECORD-TYPE to be accessed. Input and output Relation Descriptions are also generated.
- SQIBs for joining initial output relations are generated. Instructions for computations within joined relations are generated. Relation Descriptions for output relations of each of these subquery instruction blocks are generated.
- SQIBs for "vertical" computations from level cells to superior cells are generated.
- For each CDR accessed SQIBs are instantiated into canonical Subquery Control Blocks (SQCBs).

- Canonical SQCBs are connected to form the Query Execution Graph (QEG). The QEG is a partially ordered network, with SQCBs as nodes, and the directed links are descriptors of files to be passed between subqueries.

3.5.2 INTERFACE SPECIFICATIONS

3.5.2.1 Intersubsystem Data Flow Diagram

Figure 3.5.1 shows the data flows which exist between this and other Subsystems.

3.5.2.2 Subsystem Events

Figure 3.5.2 shows the major events which occur within this Subsystem.

3.5.3 TECHNICAL REQUIREMENTS

3.5.3.1 Actions

The DECOMPOSE-QUERY-INTO-CANONICAL-SUBQUERIES Subsystem translates the fully validated intermediate-query-body in the QCB into canonical subqueries. Each canonical subquery is stored in the QCB, and consists of the following components of the QCB:

- A SQCB header attached to a SQIB;
- One or more input file descriptions and an output-file description attached to the SQCB-header;

Subsystem S: DECOMPOSE-QUERY-INTO-SUBQUERIES

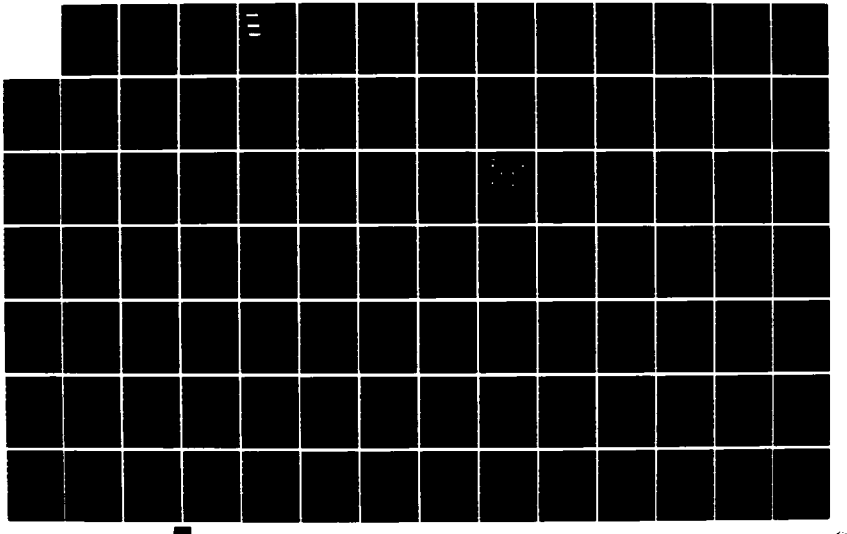
In	Data Item	Source
I1	QCB	Query Monitoring & Control /D20
I2	CDRs	IRDS

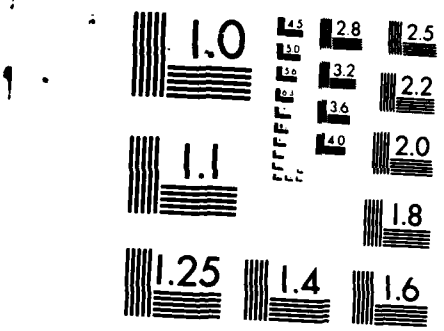
Out	Data Item	Destination
O1	QCB	Query Monitoring & Control /D20

Figure 3.5.1

AD-A173 344 DESIGN CONCEPTS FOR DATA BASE UTILITIES(U) AOB SYSTEMS 3/5
CORP HARVARD MA APR 86 RADC-TR-86-48 F30602-84-C-0011

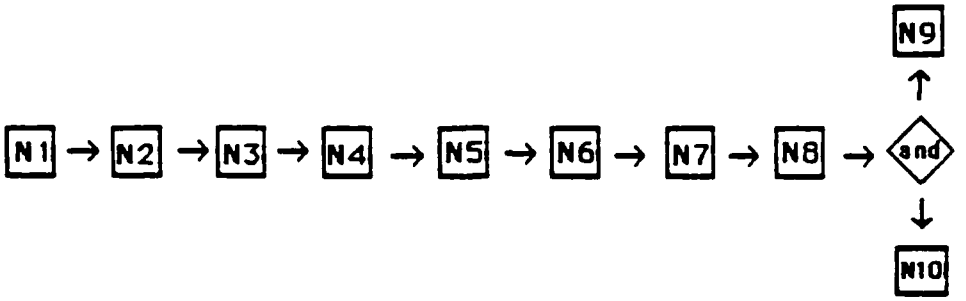
UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Subsystem 5: DECOMPOSE-QUERY-INTO-SUBQUERIES



Normal	Event Description
N1	CDR-RECORD-TYPES to be accessed are determined
N2	Canonical Subquery Instruction Blocks (SQIBs) built for each CDR-RECORD-TYPE to be accessed
N3	SQIBs built for joining CDR-RECORD-TYPES built
N4	SQIBs built/augmented for "verticle" calculation
N5	SQIBs instatiated to produce SQCBs
N6	SQCB/SQIB for final assembly is produced
N7	QEG generated
N8	Transient data purged from QCB
N9	QCB's sent to shadow D2Qs
N10	Initial subqueries shipped to D1*s for translation

Figure 3.5.2

- And a Relation Description for each input and output relation. Each Relation Description is attached to one or more file-descriptions and one or more SQIB.

The canonical Subquery Control Block (SQCB) is the aggregate of SQCB header, corresponding SQIB, file-descriptions, and relation-descriptions. (See Figure 3.5.3.)

SQCBs are connected to form the Query Execution Graph (QEG). The QEG is a partially ordered network, with SQCBs as nodes, and the directed links are descriptors of files to be passed between subqueries.

Figure 3.5.4 provides a functional decomposition of this Subsystem:

The decimal notation is used to show the hierarchical ordering of subfunctions. (In this notation ".36" indicates a the sixth second-tier subfunction invoked by the third first-tier subfunction.)

As can be seen from this diagram, decomposition of the query into subqueries is done by analyzing the cells in the query and generating a Query Execution Graph (QEG).

The main process (0) is divided into three phases. In the first phase, (.1) the query cells are traversed hierarchically where subqueries are generated. In the second phase (.2), the QEG is completed. In the third phase (.3), all temporary data structures are purged from the QCB. These temporary structures were produced to assist in generating subqueries. Most of these structures are generated by this Subsystem, but some (the E-LIST, ER-LIST, R-LIST, CR-LIST, or C-LIST) were produced by *Subsystem* 4 (DETERMINE-DATA-ACCESSIBILITY).

As the hierarchy of cells is traversed (.1), each cell is processed individually (.11). For each cell, the corresponding selection-elements are imploded to find the CDR-RECORD-TYPEs to be accessed and the corresponding CDRs (.111). (See Algorithm 1, Section 3.5.3.7 and Note 2, Section 3.5.6.) For each of these CDR-RECORD-TYPEs, a SQIB is

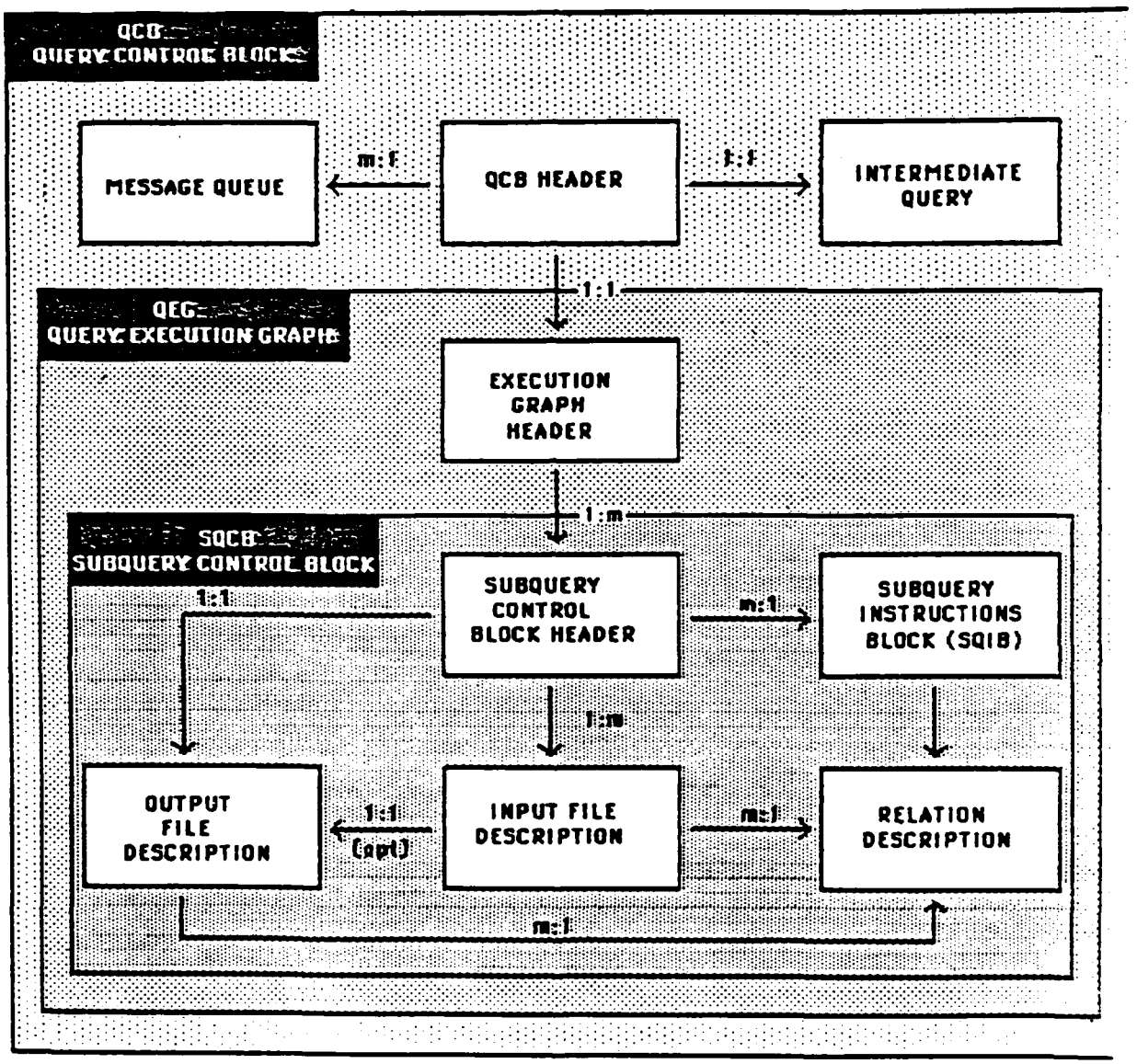

QCB
QUERY CONTROL BLOCKS
MESSAGE QUEUE
m:1
QCB HEADER
1:1
INTERMEDIATE QUERY
1:1
QEG
QUERY EXECUTION GRAPHS
EXECUTION GRAPH HEADER
1:m
SQCB
SUBQUERY CONTROL BLOCK
1:1
SUBQUERY CONTROL BLOCK HEADER
m:1
SUBQUERY INSTRUCTIONS BLOCK (SQIB)
1:m
OUTPUT FILE DESCRIPTION
1:1
(opt)
INPUT FILE DESCRIPTION
m:1
RELATION DESCRIPTION
m:1

Figure 3.5.3

Subsystem 5: DECOMPOSE-QUERY-INTO-SUBQUERIES

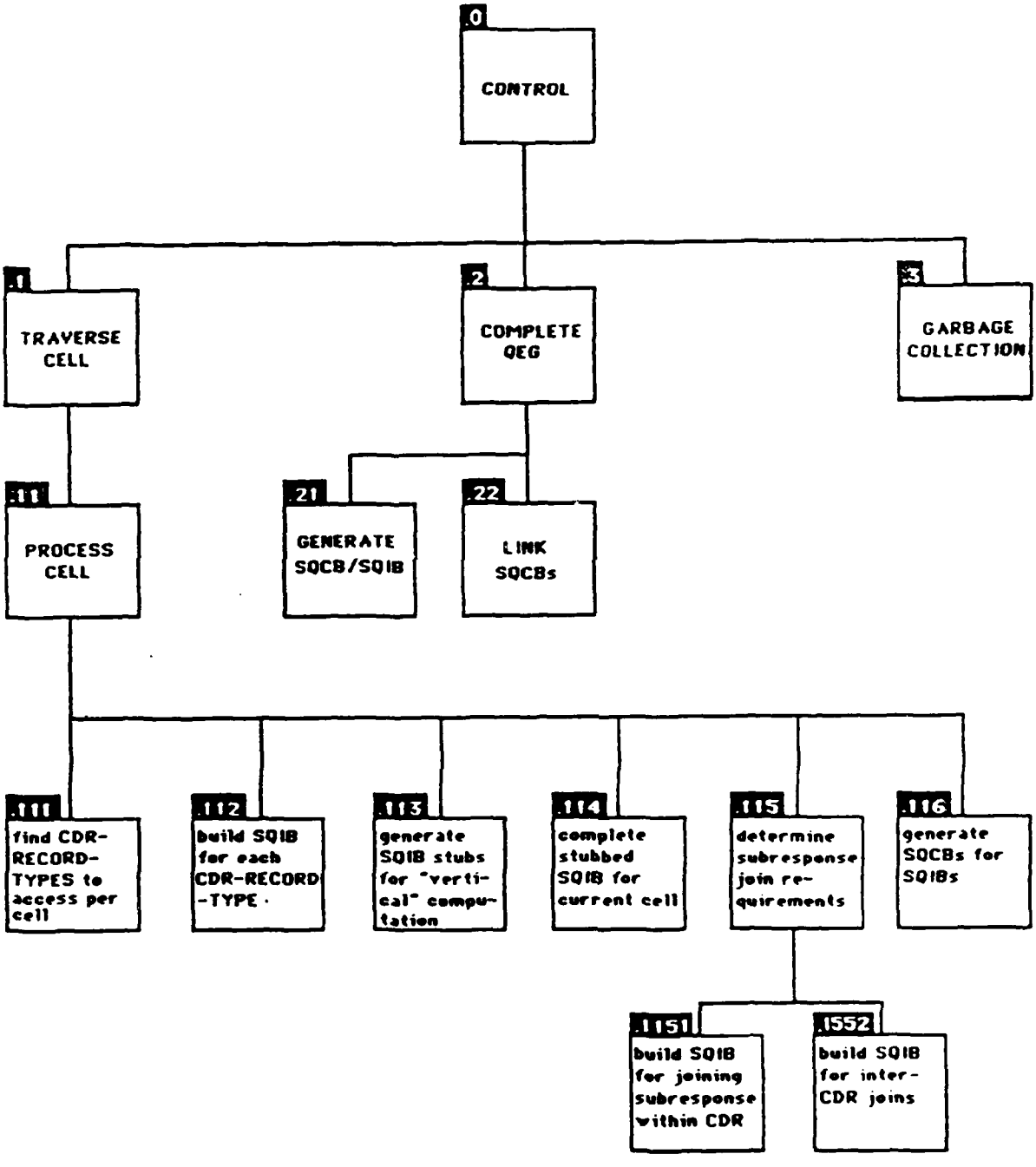


Figure 3.5.4

built (.112). (See Section 3.3.6, Note 3.) The SQIB contains the canonical form of the subquery instructions. It consists of a "SELECT" operation against the CDR-RECORD-TYPE, a "PROJECT" operation which identifies the selection-elements to be output, and the corresponding selection-criteria for each of the selection-elements. Secondary SQIBs are generated to handle "horizontal" computations. (See Section 3.5.6, No. 2). A computation-element expresses a "horizontal computation" when each form-element to the right of the equals sign is within the same cell as the computation-element.

If the cell involves "vertical" computations, a SQIB stub which identifies the selection or computation elements which must be specified for a Subordinate cell is created (.113). (A vertical computation is one in which input values are found in form-elements from a directly subordinate cell.) When a subordinate cell is expanded, its list of selection and computation elements is checked against the inputs for the computation. If there is a match, then the SQIB is completed (.114).

For each cell, each CDR is checked to determine if more than one CDR-RECORD-TYPE must be accessed (.115), if so, SQIBs which join the different CDR-RECORD-TYPEs with the CDR are generated. Cross CDR join SQIBs are then generated. (See Note 4.) Finally, for each SQIB all SQCBs for the cell are generated for each SQIB (.116) Multiple SQCBs exist whenever a SQIB may be processed against multiple CDRs. These SQCBs are chained together via input and output file descriptions.

After the cell hierarchy is completely traversed, and SQCBs are generated for subqueries and accumulated intermediate subresponses, a final SQCB is generated to bring all subresponses together and sort the final response (.2).

Finally, all intermediate working storage and data structures are purged (.3), and the QCB is returned to the QUERY-MONITORING-AND-CONTROL Subsystem at the D2Q/C.

3.5.3.2 Intrasubsystem Data Flows

Because of the specificity and complexity of this Subsystem, the numbering scheme in Figure 3.5.4 will be used to identify individual data flows. The functional decomposition in outline form is repeated here for ease of reference.

```
0     Decompose Query into Canonical Subqueries (Control)
 .1         Traverse Cells Hierarchically
 .11             Process Cell
 .111                 Find CDR-RECORD-TYPEs to access per cell
 .112                 Build SQIB for each CDR-RECORD-TYPE
 .113                 Generate SQIB stubs for "vertical" computations
 .114                 Complete stubbed SQIB for current cell
 .115                 Determine subresponse join requirements
 .1151                    Build SQIB for joining subresponses within CDR
 .1152                    Build SQIB for inter-CDR joins
 .116                 Generate SQCBs for SQIBs
 .2         Complete QEG
 .21             Generate SQCB/SQIB
 .22             Link SQCBs
 .3         Garbage collection
```

FROM	TO	DATA ITEMS
0	.1	QCB, E-LIST, ER-LIST, R-LIST, CR-LIST, C-LIST
0	.2	QCB
0	.3	L4, L5, E-LIST, ER-LIST, R-LIST, CR-LIST, C-LIST
.1	.11	QCB, Current Cell, Cell-key
.11	.111	QCB, Current Cell, E-LIST, ER-LIST, R-LIST, CR-LIST, C-LIST, Cell-key & Selection elements within cell
.111	.11	L4, L5
.11	.112	QCB, L4
.11	.113	QCB, Current Cell, L4, computation-elements

FROM	TO	DATA ITEMS
.11	.114	QCB, Current Cell, L4, computation-elements, SQIB-Stub
.11	.115	QCB, L5, Current Cell, Cell-key
.11	.116	QCB, L5
.115	.1151	QCB, L5, Current Cell, Cell-key
.115	.1152	QCB, L5, Current Cell, Cell-key
.2	.21	QCB
.2	.22	QCB

3.5.3.3 Tasking

One task exists for each query.

3.5.3.4 Exception Conditions

None.

3.5.3.5 Rationale for Critical Decisions

1) Except for the final response, each subresponse will consist of a single Second Normal Form relation.

   It is assumed that each subquery which accesses a data node will be translated into a local query language. Accordingly, a "lowest common denominator" of query output capability must be assumed. All query languages provide a simple "listing" facility, and this is taken to be the minimal required capability.

   If a relation is in second normal form, then no non-key attributes will repeat. In a query, each cell corresponds to a Second Normal Form relation.

Since a subquery constructs output for one of the query cells, each subresponse will be at least in Second Normal Form.

2) All computations will be performed at IDN-nodes rather than at Data Nodes.

There is a tradeoff between overall simplicity and efficiency. It is possible that this decision may generate more response traffic from the Data Node to the D1*, and more processing at the D1*.

On the other hand, given the different target DBMSs, it is possible that different DBMSs will have different standard function sets. Unless a minimal common function set is imposed on the IDN, any given function might not be handled consistently by different target DBMSs. Furthermore, one of the benefits of the IDN is that the function set is extensible. IDN-specific functions are best implemented entirely within the IDN. Finally, some computations must be performed on subresponses. Thus the IDN processors must perform computations. This approach provides for meeting the computational requirements in the most direct way, and also provides the simplest possible interface to the target DBMSs.

3.5.3.6 Open / Deferred Decisions

None.

3.5.3.7 Critical Algorithms

1. Find CDR-RECORD-TYPES to Access

FIND-CDR-RECORD-TYPES-TO-ACCESS-PER-CDR PROCEDURE:

BEGIN PROCEDURE.

```
Build the lists:

     L1:  A list of selection-elements which are key.
     L2:  A list of CDR-KEYs.
     L3:  A secondary list consisting of CDR-KEY/CDR-ELEMENTs.
     L4:  A list consisting of Cell/CDR-KEY/CDR-RECORD-TYPE.
     L5:  A list consisting of Cell/CDR-RECORD-TYPE/DPP-ACCESS-
          FLAG/CDR.

/* L1, L2, L3 are cleared at the end of this proc */
/* L4 and L5 are retained */

/* See special data requirements */

For each selection-element within the cell:

     Find all CDR-KEYs which contain the selection-element, if any.
     If the CDR-ELEMENT belongs to a CDR-KEY, save it in L1.

     For each CDR-KEY containing the selection-element:

          If the CDR-KEY is not already in L2:

               Save the CDR-KEY.
               Find all CDR-ELEMENTS with the current CDR-KEY.
               Save each CDR-KEY/CDR-ELEMENT in L3.

/* At this point, L1 defines the "maximal" key (Section 3.5.6, Note 2 )
*/

     For each CDR-KEY in L2:

          Loop through L3 entries for the current CDR-KEY.
          If any corresponding CDR-ELEMENT in L3 is not in L1:
```

```
            Remove the current CDR-KEY from L2.
            Remove all corresponding entries in L3.

/*    At this point, L2 contains only the maximal key and subkeys
      See Section 3.5.6, Note 2. */

For each CDR-KEY remaining in L2:

      Identify all corresponding CDR-RECORD-TYPEs which are in the
      R-LIST.  /* R-LIST passed in QCB from Subsystem 4 */

      For each corresponding CDR-RECORD-TYPE:

            Using the ER-LIST (from Subsystem 4), identify all CDR-
            ELEMENTS for the CDR-RECORD-TYPE.

            If the cell has any non-key selection-element which is
            within the CDR-RECORD-TYPE:

                  Save in L4, the current cell, the CDR-KEY, and
                  CDR-RECORD-TYPE.

/* At this point, all CDR-RECORD-TYPEs which have to be
accessed are known.  It is now necessary to determine which CDRs
are to be accessed for each CDR-RECORD-TYPE and to identify if DPP
causes any modifications to the selection criteria.  (See Section
3.5.6, Note 3.)  If DPP modifications of selection criteria are
required, then one SQIB will be generated per CDR/CDR-RECORD-TYPE.
Otherwise, only one SQIB will be generated, and it will be shared
by all SQCBs which access the CDR-RECORD-TYPE. */

For each CDR-RECORD-TYPE in L4:

      /*    DPP-FLAG identifies that DPP is defined for at least one
            component element of the record-type. */
```

```
Set DPP-FLAG = "N".
If the selection-element VIRTUAL-KEY is not specified in the
cell:

/*   If VIRTUAL-KEY is not specified in the cell then DPP
     may cause modifications of selection criteria in order to
     keep the cell's output relation a Second Normal Form
     relation.
     If VIRTUAL-KEY is specified, then DPP is overridden and
     and the cell's output relation is assured of being Second
     Normal Form.
*/
     Find corresponding CDR-ELEMENTs using the ER-LIST.

     Until all corresponding CDR-ELEMENTs are examined or
     DPP-FLAG = "Y":

          If the corresponding CDR-ELEMENT is specified in the
          cell:

               If the CDR-ELEMENT has a VALUE-SET which is
               related to a CDR:

                    Set DPP-FLAG = "Y".

          Point to next corresponding CDR-ELEMENT.

     Find all RC-LIST entries for the given CDR-RECORD-TYPE.
     For each such entry:

          Build a L5 list entry, setting DPP-ACCESS-FLAG=DPP-FLAG.

/* It is now known which CDR-RECORD-TYPEs must be accessed from
which CDRs and which accesses require modifications to selection
criteria.
*/
```

Clear L1, L2, L3.

END PROCEDURE.

3.5.4 DATA REQUIREMENTS

3.5.4.1 Explanation of Major Data Flows

Query Control Block (QCB)

The QCB has been defined previously.

3.5.4.2 Special Data Requirements

The various lists identified here and those passed from Subsystem 4 may be best transformed into a single matrix.

3.5.5 DATABASE REQUIREMENTS

3.5.5.1 Network Metadata

None.

3.5.5.2 Data Metadata

The following entity-types are used:

CDR
CDR-RECORD-TYPE
CDR-KEY
CDR-ELEMENT
VALUE-SET

The following relationship-types are used:

CDR-SET-OF-CDR-RECORD-TYPE
CDR-RECORD-TYPE-SET-OF-CDR-ELEMENT
CDR-RECORD-TYPE-HAS-CDR-KEY
CDR-KEY-SET-OF-CDR-ELEMENT
CDR-ELEMENT-HAS-VALUE-SET
VALUE-SET-IN-CDR

3.5.5.3 Other

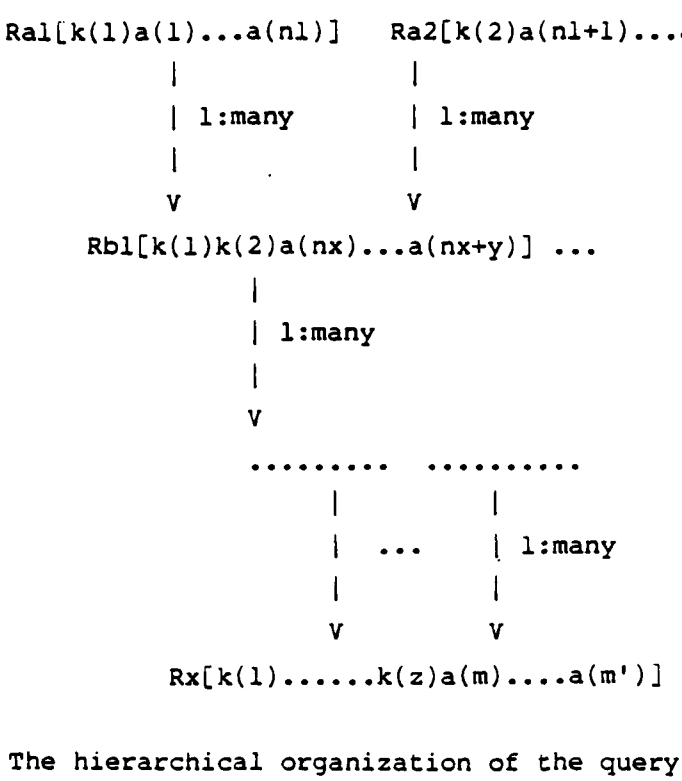
None.

3.5.6 NOTES

1. The lists produced by Subsystem 4 (E-LIST, ER-LIST, R-LIST, CR-LIST, and C-LIST) may be used directly, rather than accessing the dictionary again. Alternatively, these lists might be transformed into a matrix which could be used to identify CDR-ELEMENT / CDR and CDR / query-cell relationships directly. Other alternatives are also possible. The specification of the actual mechanism is deferred to the detailed design.

2. The decomposition of queries assumes several constraints on the structure of the query and the CDRs. In particular, each CDR is assumed to be composed of CDR-RECORD-TYPES which are at least Third Normal Form relations. Also, the hierarchical structure of the query cells assumes that the selection elements of each cell have a many-to-one relationship with the selection-elements of the directly superior cell. Each cell is also assumed to contain a given selection element only once. In order to achieve this, we require that for each cell, a Second Normal Form relation can be constructed, which consists of the selection elements within the cell, and "key" selection elements in superior cells. By a "key" for an element we mean a set of data elements which functionally determine the given element. For the root cell, all key elements are selection elements within the root cell. For subordinate cells, the key elements are selection elements in the given and superior cells. (For non-root cells, at least one key element will be in the directly superior cell.)

   In decomposing the query, the query's selection elements an be projected onto each CDR corresponding to a database which must be accessed. Since each CDR is assumed to be composed of at least Third Normal Form relations, a partial ordering can be constructed among the relations in the CDR, where "R(i) < R(j)" is interpreted to mean that R(i) has a one to many relationship with R(j). This situation can be pictured as follows, where Rx's indicate relations, k(y)'s indicate key elements, and a(z)'s indicate non-key elements:



The hierarchical organization of the query implies several things:

First, the root cell of the query will identify a certain number of key elements. There will be one and only relation in the CDR with these key elements. The other selection elements within the root cell must either be in this relation, or some superior relation in the partial ordering (i.e., in a relation which is both connected and above in the diagram.)

Next, as we proceed down any branch in the hierarchy of cells, each subordinate cell will have a set of key elements which consist of key elements of the superior cell, and one or more new key elements. This will then correspond to another relation in the CDR. Each selection element in the cell will then be found in this this relation, or a superior relation which is not in the subtree of relations which are inferior to or equal to the relations which correspond to any superior cell(s). For example, suppose relation Rb1 in the above diagram corresponds to a a given query cell C, and

relation Ra1 corresponds to C's superior cell. Then C may contain selection elements found in relations Rb1 and Ra2, but not Ra1.

The relation which corresponds to the given cell is called the **maximal relation** in the CDR for that cell. (The term "maximal" is chosen because it is the highest relation in the partial ordering which may contain selection elements in the given cell.) Likewise the key for this relation is called the **maximal key.** (The key is maximal in the sense that any selection element in that cell which exists in that CDR will be found in a relation whose key is a subkey of the key for the maximal relation.)

Note that the term "maximal relation" suggests that there may be more than one maximum. Indeed this is the case. There are situations where several relations can exist in the same database, each with the same key, but for any given set of key values, only one relation will have a corresponding tuple. This is equivalent to having a single Third Normal Form relation, but always having certain disjoint sets of attributes null for any key value. Although either representation may be proper from a strict relational database point of view, the IDN requires that multiple CDR-RECORD-TYPES be defined in these cases. A query will then be defined using **alternating cells**, where each cell corresponds to precisely one of the maximal relations.

3. Normally a SQIB is generated on a CDR-RECORD-TYPE basis. However, Delegated Production Policy (DPP) may force SQIBs to be customized for individual CDRs. In accessing relations from different CDRs, DPP may require changing the selection criteria for certain CDR-ELEMENTs. Consider the following example:

   Suppose R is a CDR-RECORD-TYPE, with CDR-ELEMENTs a(1), ..., a(m). Suppose R is common to CDRs D1 and D2. For CDR-ELEMENT a(i), DPP states that values v1 through v2 are to be found in D1, and values v3 through v4 are to be found in D2. Now D1 and D2 may contain overlapping data. If the selection criterion for a(i) is

implicitly "all" or covers values from both CDRs, then the selection criteria must be changed for each CDR to be accessed.

4. Joining of relations across CDRs is done on the basis of common key values. Therefore, if DPP were to partition the key values across several CDRs, no joining of relations across those CDRs is possible.

This inability to join relations is not a problem provided that each CDR had the same set of CDR-RECORD-TYPEs. In this case, the CDRs would constitute fragments of a distributed database.

However, suppose that a CDR-KEY K consists of CDR-ELEMENTs k(1), k(2), k(3), and suppose K is a maximal key for a query cell C. Also assume CDR1 contains the CDR-RECORD-TYPE with key K, and CDR2 and

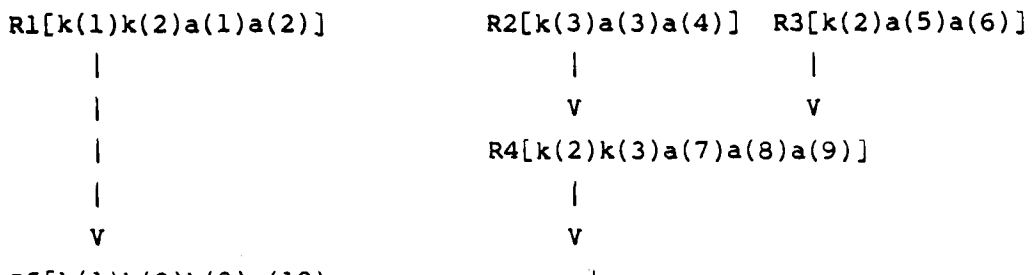
CDR3 contains the CDR-KEY K' consisting of just k(1). If DPP identifies CDR2 and CDR3 as the only sources for k(1), then CDR1 cannot be accessed.

There are two solutions to this problem:

- Do not specify DPP for key elements.
- For key elements, associate a given VALUE-SET with multiple CDRs.

The former is probably not a reasonable solution. In the latter case, it is entirely possible that the partitioning of values for a given key element k(i) may be different when different sets of CDRs contain k(i) in combination with other different key elements.

This situation can be illustrated by the following diagram:



Suppose, in this case, that the data was partitioned as follows:

R1 and R5 in CDR1, CDR2

and

R2, R3, and R4 in CDR3, CDR4, CDR5, and CDR6.

Then it is possible to have a key element, say k(2), which has the following partitioning from DPP:

CDR1	1 through 150
CDR5	151 through 300
CDR2	1 through 100
CDR3	101 through 200
CDR4	201 through 300

(This page intentionally left blank)

3.6 SUBSYSTEM 6: IMPLEMENT-EXECUTION-STRATEGY

3.6.1 OVERVIEW

Subsystem 6, the IMPLEMENT-EXECUTION-STRATEGY Subsystem, uses the QEG to package canonical SQCBs into canonical subquery transactions, and maintains the QEG for the duration of the query. If multiple subqueries are to execute at a given Data Node, then they are shipped to the corresponding D1* as a series of canonical subqueries.

During the execution of a query, this Subsystem is reactivated as subqueries change state. Subquery state transitions are:

- Subquery assigned to node.
- Subquery accepted.
- Subquery initiated.
- Subquery completed.
- Subresponse received.
- Subquery terminated (i.e., abnormal termination).
- Subquery backout.
- Subquery stopped (i.e., put in wait state).
- Subquery restarted.

This Subsystem accepts the subquery state transition messages, updates the subquery status indicators in the QEG, and decides when subsequent subqueries are to be initiated. It then packages those SQCBs into canonical subquery transactions and ships them to their appropriate destinations.

This Subsystem works in coordination with Subsystem 7 (the QUERY-MONITORING-AND-CONTROL Subsystem) and Subsystem 12 (the IMPLEMENT-SUBRESPONSE-STRATEGY Subsystem).

3.6.2 INTERFACE SPECIFICATIONS

3.6.2.1 Intersubsystem Data Flow Diagram

Figure 3.6.1 shows the data flows which exist between this and other Subsystems.

3.6.2.2 Subsystem Events

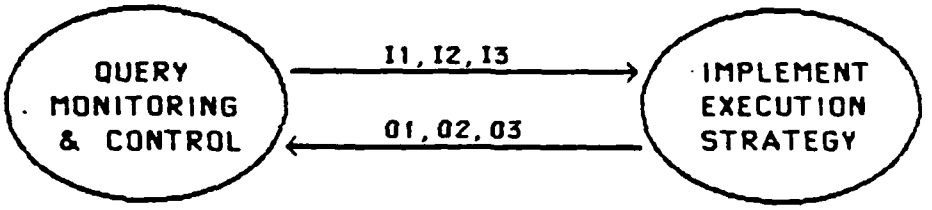
Figure 3.6.2 shows the major events which occur within the Subsystems.

3.6.3 TECHNICAL REQUIREMENTS

3.6.3.1 Actions

Upon initiation, this Subsystem analyzes the QEG and builds a queue of pending subqueries. As subquery transactions are built, the names of the

Subsystem 6: IMPLEMENT-EXECUTION-STRATEGY

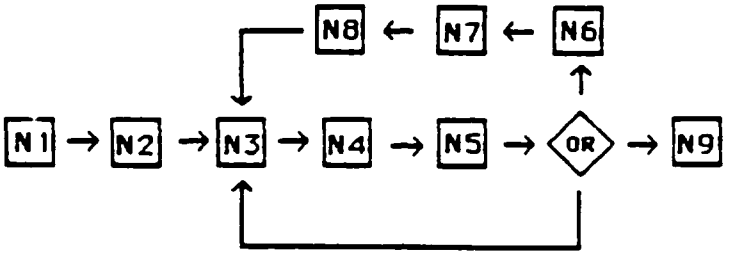


In	Data Item	Source Subsystem
I1	QCB	Query Monitoring & Control
I2	Subquery Status Message	Query Monitoring & Control
I3	Processor Role Change Msg	Query Monitoring & Control

Out	Data Item	Destination Subsystem
O1	Updated QCB	Query Monitoring & Control
O2	Canonical Subquery	Query Monitoring & Control
O3	Query Completion Message	Query Monitoring & Control

Figure 3.6.1

Subsystem 6: IMPLEMENT-EXECUTION-STRATEGY



Normal	Event Description
N1	Initial QEG received from QMC
N2	Initial Subqueries forwarded to QMC for shipping to DI*s
N3	Subsystem waits for status messages
N4	Status message received
N5	QEG updated with status information
N6	Node assigned to Subquery
N7	Canonical Subquery Transaction Created
N8	Canonical Subquery forwarded to QMC
N9	QEG shows all subqueries completed

Figure 3.6.2

subqueries are removed from the pending subqueries queue. All subqueries which have no prerequisites are built and shipped to their appropriate destinations.

The subsystem now waits for status messages. As status messages are received, the QEG is updated. The pending subquery queue is examined to determine if any pending subquery's prerequisites have been fulfilled. If so, a canonical subquery transaction is generated and shipped.

Some messages may call for backing out one or more subqueries. In this case, the QEG is updated to indicate that the query has not been executed, and the subquery is added back to the pending queue.

Processing proceeds until either all subqueries have completed execution or until the QUERY-MONITORING-AND-CONTROL Subsystem terminates the query. Upon completion of all subqueries, this Subsystem notifies the QUERY-MONITORING-AND-CONTROL Subsystem that the query has completed processing.

3.6.3.2 Intrasubsystem Data Flows

Figure 3.6.3 shows the major processing components within the Subsystem and the data flows that exist between them.

3.6.3.3 Tasking

In this Subsystem one task exists for each active query at each D2Q assigned to the query. At the D2Q/C, this Subsystem will create all subqueries and maintain status information in the QEG. At each D2Q/S, this Subsystem will maintain the status information in the QEG. The task will be terminated whenever QUERY-MONITORING-AND-CONTROL Subsystem determines that the query is completed or to be terminated.

Subsystem 6: IMPLEMENT-EXECUTION-STRATEGY

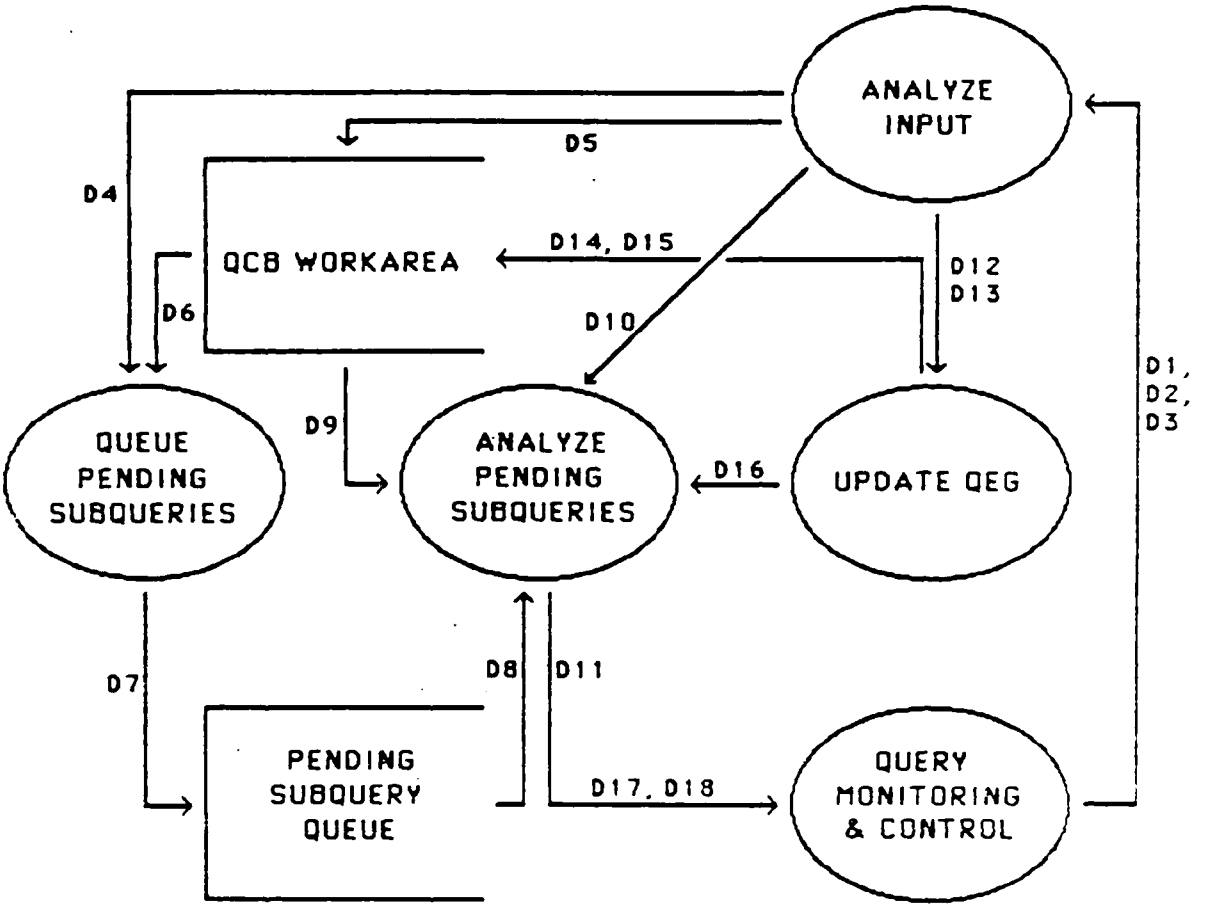


Figure 3.6.3

Subsystem 6: IMPLEMENT-EXECUTION-STRATEGY

Data Flow	Data Item	Source Function / Store	Destination Functional / Store
D1	QCB	QMC	Analyze Input
D2	Subquery Status Message	QMC	Analyze Input
D3	Processor Role Change Message	QMC	Analyze Input
D4	Build Initial Pending Queue Message	Analyze Input	Build Pending Subquery Queue
D5	QCB	Analyze Input	QCB Workarea
D6	QEG	QCB Workarea	Build Pending Subquery Queue
D7	Pending Subquery Header	Build Pending Suquery Queue	Pending Subquery Queue
D8	Pending Subquery Header	Pending Subquery Queue	Analyze Pending Subqueries
D9	QEG	QCB Workarea	Analyze Pending Subqueries
D10	Start Analysis Message	Analyze Input	Analyze Pending Subqueries
D11	Canonical Subquery	Analyze Pending Subqueries	QMC
D12	Subquery Status Message	Analyze Input	Update QCB
D13	Processor Role Change Message	Analyze Input	Update QCB
D14	Subquery Status Change	Update QCB	QCB Workarea
D15	Processor Reassignment	Update QCB	QCB Workarea
D16	Start Analysis Message	Update QCB	Analyze Pending Subqueries
D17	Updated QCB	Analyze Pending Subqueries	QMC
D18	Query Completion Message	Analyze Pending Subqueries	QMC

Figure 3.6.3

3.6.3.4 Exception Conditions

1. If the processor at which this Subsystem is running is upgraded (from shadow to controlling node) or downgraded, the QUERY-MONITORING-AND-CONTROL Subsystem will notify the IMPLEMENT-EXECUTION-STRATEGY Subsystem of this fact. If the processor's role for a query has been upgraded, then the task which has been maintaining the QEG as the node will start building Canonical Subquery transactions. Likewise, if the processor's role has been downgraded, the processor will stop building these transactions. In the downgrade situation, this Subsystem will complete the building of any subqueries which it had already started.

3.6.3.5 Rationale for Critical Decisions

None.

3.6.3.6 Open / Deferred Decisions

None.

3.6.3.7 Critical Algorithms

None.

3.6.4 DATA REQUIREMENTS

3.6.4.1 Explanation of Major Data Flows

Processor Role Change Message

This transaction identifies the primary processor which has been downgraded to a shadow role, and the newly designated primary processor.

Query Execution Graph (QEG)

The QEG has already been defined in detail.

Subquery Status Messages

The Subquery Status Messages are as follows:

- Subquery-assigned-node Message.
- Subresponse-received Message.
- Subquery-accepted Message.
- Subquery-initiated Message.
- Subquery-completed Message.
- Subquery-terminated (i.e., abnormal termination) Message.
- Subquery-backout Message.
- Subquery-stopped (i.e., put in wait state) Message.

- Subquery-restarted Message.

Each Message consists of the subquery identifier and the new subquery state (as identified in the Overview). In addition, the Subquery-assigned-node Message identifies the assigned-node, and the Subresponse-received Message identifies the input-file which has been received.

The Subquery-assigned-node Message is primarily an internal message passed between the QUERY-MONITORING-AND-CONTROL (QMC) Subsystem and this Subsystem. It is, however, also passed from the QMC at the D2Q/C to each D2Q/S.

The Subquery-backout Message is a directive passed from QMC to this Subsystem. It also is passed from the D2Q/C to each corresponding D2Q/S.

All the other messages are initiated at the node assigned to the given subquery.

Canonical Subquery

The Canonical Subquery consists of:

- A transaction header;
- Subquery control block header;
- The corresponding Subquery Instruction Block (SQIB);
- A File Description for each input file;
- A File Description for each output file;
- The corresponding Relation Description for each of the above File Descriptions.

3.6.4.2 Special Data Requirements

None.

3.6.5 DATABASE REQUIREMENTS

3.6.5.1 Network Metadata

None. The network metadata used by this Subsystem is either embedded in the QCB, and remains unchanged, or comes in a processor role change message.

3.6.5.2 Data Metadata

None.

3.6.5.3 Other

None.

3.6.6 NOTES

None.

(This page intentionally left blank)

3.7 SUBSYSTEM 7: QUERY-MONITORING-AND-CONTROL

3.7.1 OVERVIEW

Subsystem 7, the QUERY-MONITORING-AND-CONTROL Subsystem (QMC), is responsible for coordinating all phases of query processing, monitoring the progress of the query, and responding to error conditions which might arise. In order to accomplish this effectively, QMC is distributed across each of the processor types rather than residing at a single processor type or even a single processor. A "QMC Process" must therefore be resident at each of the D1, D2Q, and D1* Nodes. Such a process must also run at any D3Q Nodes which are active in the IDN. The D2Q processor, and the corresponding QMC Process, which validates a given query and decomposes it into subqueries, is designated the "Controlling D2Q" (D2Q/C) Node with respect to the query. A D2Q Node which receives a copy of the control information for the query for backup purposes is designated a "Shadow D2Q" (D2Q/S) Node.

The communication which occurs among the coordinated QMC Processes is based entirely upon the Query and Subquery Control Blocks (QCBs and SQCBs) which are passed from one node to another in the course of processing queries and responses. Messages are passed between QMC Processes in order to update control blocks, as necessary, to reflect progress in the processing of a query. In addition to the QCBs and SQCBs, information for monitoring and controlling the execution of subqueries is:

- Provided to QMC by the IMPLEMENT-EXECUTION-STRATEGY Subsystem.
- Stored in the form of a Query Execution Graph at the D2Q/C and D2Q/Ss.

- Executed by QMC in order to process the subqueries.

QMC must also be "programmed" to handle the composition of the subresponses for the query. The decisions which must be made for subresponse composition cannot be made in advance since they are based on the relative volumes of the subresponses. QMC must coordinate this process by:

- Passing subresponse volumes to the IMPLEMENT-COMPOSITION-STRATEGY Subsystem at the D2Q/C Node.

- Receiving composition instructions from the IMPLEMENT-COMPOSITION-STRATEGY Subsystem in the form of a Response Composition Program (SEMI-JOIN operations).

- Communicating these instructions to the ASSEMBLE-COMPOSITE-RESPONSE Subsystem at the D1* Nodes which hold the subresponses.

The QUERY-MONITORING-AND-CONTROL Subsystem must also deal with various contingencies which may arise in the network, such as the varying traffic loads and the failure of one or more nodes in the network. These responsibilities are met by monitoring status information for the network, including both the processors and communications lines. This status information is provided to QMC, and to all of the other Subsystems, by special "Network Information" (NI) processors so that queries and control blocks may be properly routed.

Based upon the information available to it, QMC will:

- Selectively purge lower precedence messages from processing queues in order accelerate processing of higher precedence messages when traffic is high.

- Enforce the quotas that have been established for subresponse output and for overall response output for a given query.

- Initiate restart procedures and recovery of queries which have been terminated prematurely because of any capacity limitations in the IDN.

- Insure that queries which are terminated as a result of security violations are logged in the IDN audit trail, and notifications are sent to the security administrators before the queries are purged from the network.

3.7.2 INTERFACE SPECIFICATIONS

3.7.2.1 Intersubsystem Data Flows

Figure 3.7.1 shows the data flows between this and other Subsystems. Instead of a detailed data flow diagram showing all of the individual flows, an overall view of the interaction between QMC at the D1, D2Q, and D1* processors with the other Subsystems is provided. The diagram also indicates the types of the processors at which the various Subsystems reside. The Subsystem 7 Data Flow Table, which illustrates a detailed breakdown of the data flows to (input) and from (output) QMC and the other Subsystems, follows the diagram. The letters which identify the general interactions in the diagram correspond to groups of data flows in the table. The following notation is used to differentiate QMC at the three primary node types:

- QMC|D1 - QMC at a D1 Node.
- QMC|D2 - QMC at a D2Q Node.
- QMC|D1* - QMC at a D1* Node.

Subsystem 7: QUERY-MONITORING-AND-CONTROL

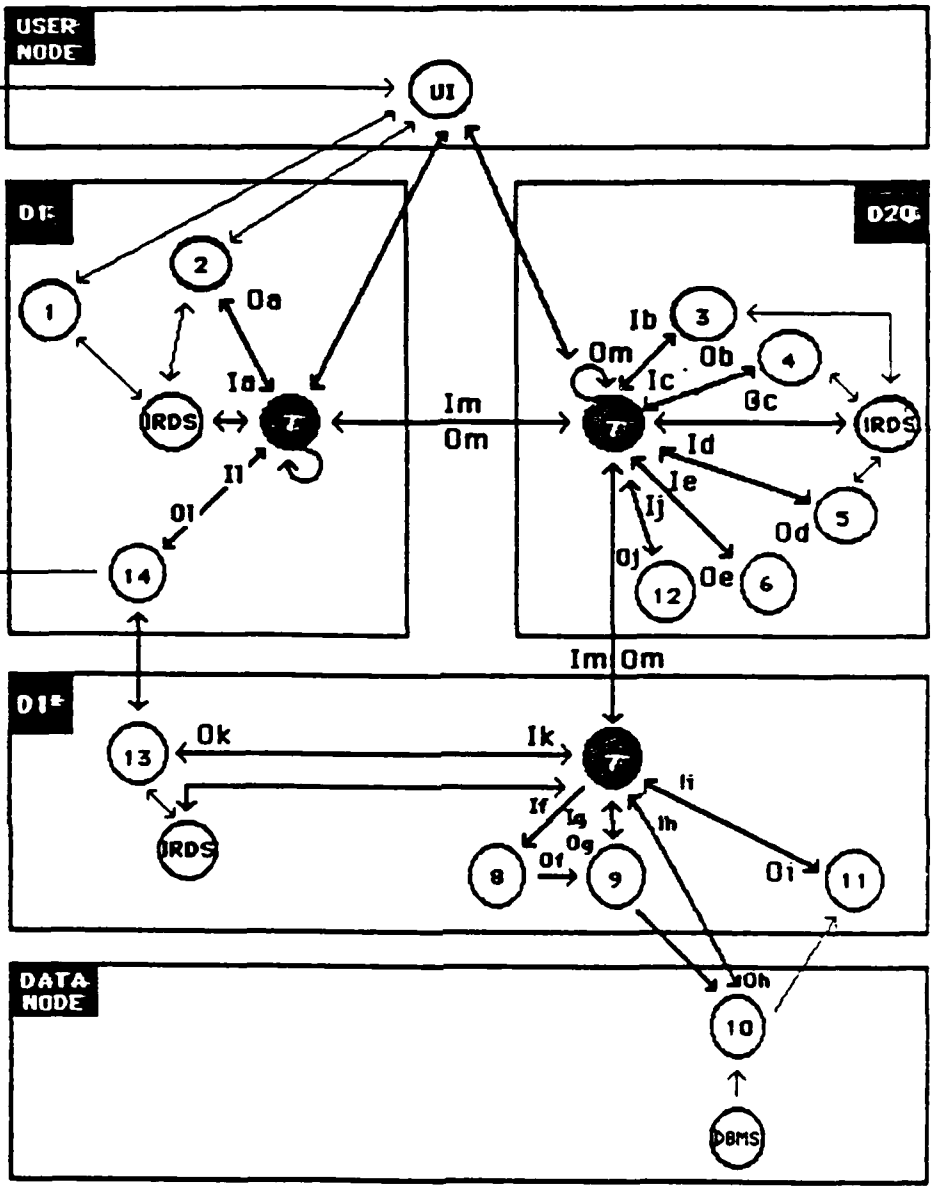


Figure 3.7.1

INPUT	DATA ITEM	SOURCE SUBSYSTEM
QMC|D1		
Ia-1	destination-request	the VALIDATE-TOKENIZED-QUERY Subsystem
Ia-2	intermediate-query	the VALIDATE-TOKENIZED-QUERY Subsystem
QMC|D2Q		
Ib-1	QCB	the VALIDATE-INTERMEDIATE-QUERY Subsystem
Ic-1	QCB	the DETERMINE-DATA-ACCESSIBILITY Subsystem
Id-1	QCB/SQCBs	the DECOMPOSE-INTO-SUBQUERIES Subsystem
Ie-1	updated-QEG	the IMPLEMENT-EXECUTION-STRATEGY Subsystem
Ie-2	SQCBs	the IMPLEMENT-EXECUTION-STRATEGY Subsystem
Ie-3	query-completion-message	the IMPLEMENT-EXECUTION-STRATEGY Subsystem
QMC|D1*		
If-1	SQCBs	the RECEIVE-SUBQUERIES Subsystem
Ig-1	SQCBs	the TRANSLATE-TO-NDRL Subsystem
Ih-1	subquery-execution-error	the EXTRACT-AND-COMPUTE-DATA Subsystem
Ih-2	subresponse-size	the EXTRACT-AND-COMPUTE-DATA Subsystem
Ih-3	subresponse-over-quota	the EXTRACT-AND-COMPUTE-DATA Subsystem
Ii-1	assembly-node-id-request	the PREPARE-OUTPUT-FOR-NETWORK Subsystem

INPUT	DATA ITEM	SOURCE SUBSYSTEM
QMC|D2Q		
Ij-1	response-composition-program	the IMPLEMENT-COMPOSITION-STRATEGY Subsystem
QMC|D1*		
Ik-1	subresponse-status-data	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem
Ik-2	output-transmission-error	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem
Ik-3	response-composition-error	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem
Ik-4	response-composition-complete	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem
QMC|D1		
Il-1	response-delivered-to-user	the DELIVER-COMPOSITE-RESPONSE Subsystem
Im-1	purge-QCB-SQCBs-QEG	the QUERY-MONITORING-AND-CONTROL Subsystem at the D1 Node

OUTPUT	DATA ITEM	DESTINATION SUBSYSTEM
QMC|D1		
Oa-1	destination-data	the VALIDATE-TOKENIZED-QUERY Subsystem
QMC|D2Q		
Ob-1	QCB	the VALIDATE-INTERMEDIATE-QUERY Subsystem
Oc-1	QCB	the DETERMINE-DATA-ACCESSIBILITY Subsystem

OUTPUT	DATA ITEM	DESTINATION SUBSYSTEM
Od-1	QCB	the DECOMPOSE-INTO-SUBQUERIES Subsystem
Oe-1	QCB/SQCBs	the IMPLEMENT-EXECUTION-STRATEGY Subsystem
Oe-2	QEG	the IMPLEMENT-EXECUTION-STRATEGY Subsystem
Oe-3	subquery-status-messages	the IMPLEMENT-EXECUTION-STRATEGY Subsystem
Oe-4	processor-role-change-msgs	the IMPLEMENT-EXECUTION-STRATEGY Subsystem
QMC|D1*		
Of-1	SQCBs	the RECEIVE-SUBQUERIES Subsystem
Og-1	SQCBs	the TRANSLATE-TO-NDRL Subsystem
Oh-1	abort-subquery-processing	the EXTRACT-AND-COMPUTE-DATA Subsystem
Oi-1	assembly-node-id	the PREPARE-OUTPUT-FOR-NETWORK Subsystem
Oj-1	subresponse-size	the IMPLEMENT-COMPOSITION-STRATEGY Subsystem
QMC|D2Q		
Ok-1	perform-SEMI-JOIN-select	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem
Ok-2	perform-SEMI-JOIN-join	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem
Ok-3	perform-SEMI-JOIN-append	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem
Ok-4	send-output-directive	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem
Ok-5	cancel-output-directive	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem

OUTPUT	DATA ITEM	DESTINATION SUBSYSTEM
Ok-6	subresponse-status-request	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem
Ok-7	transmit-response-to-user	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem
QMC|D1		
O1-1	hold/transmit response	the DELIVER-COMPOSITE-RESPONSE Subsystem
Om-1	purge-SQCBs	the QUERY-MONITORING-AND-CONTROL Subsystem at the D1* Node
Om-2	purge-QCB-SQCBs	the QUERY-MONITORING-AND-CONTROL Subsystem at the D2Q Node
Om-3	purge-QEG	the QUERY-MONITORING-AND-CONTROL Subsystem at the D2Q Node
Om-4	purge-QCB	the QUERY-MONITORING-AND-CONTROL Subsystem at the D1 Node
Om-5	final-output-held-at-*node-msg*	the QUERY-MONITORING-AND-CONTROL Subsystem at the D1 Node
Om-6	final-output-purged-from-*node-msg*	the QUERY-MONITORING-AND-CONTROL Subsystem at the D1 Node
QMC|D2Q		
Om-7	restart-query-directive	the QUERY-MONITORING-AND-CONTROL Subsystem at the D1, D2Q, and D1* Nodes

3.7.2.2 Subsystem Events

The event diagram for this Subsystem is shown in Figure 3.7.2.

Subsystem 7: QUERY-MONITORING-AND-CONTROL

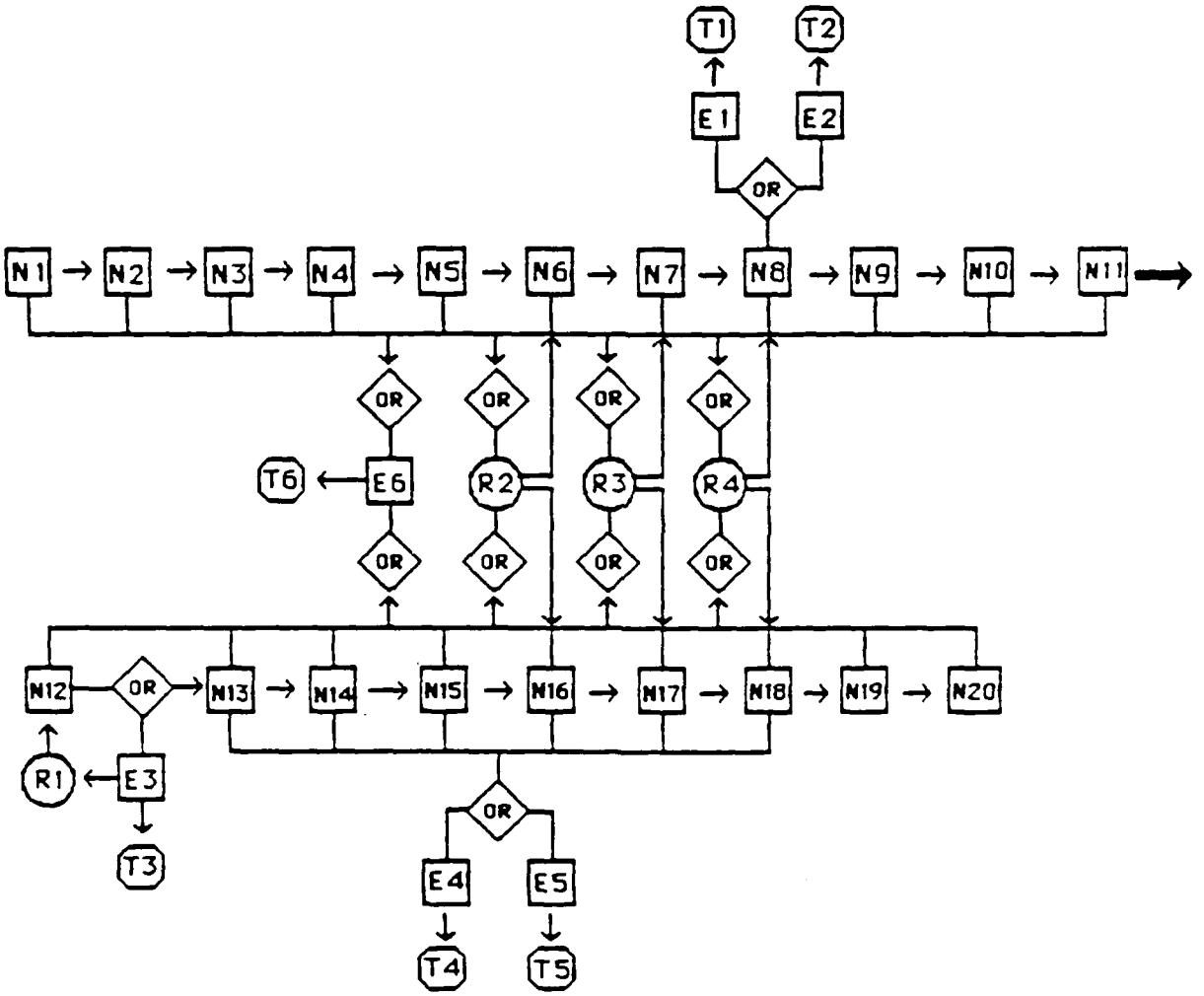


Figure 3.7.2

NORMAL	EVENT DESCRIPTION
QMC|D1	
N1	Determine D2Q destination-id for intermediate query.
N2	Pass QCB for intermediate query to D2Q for validation.
QMC|D2Q	
N3	Pass QCB for validated query to the DETERMINE-DATA-ACCESSIBILITY Subsystem.
N4	Monitor and Control: the DECOMPOSE-INTO-SUBQUERIES Subsystem.
N5	Monitor and Control: the IMPLEMENT-EXECUTION-STRATEGY Subsystem.
QMC|D1*	
N6	Monitor and Control: Receive Subqueries at D1*.
N7	Monitor and Control: the TRANSLATE-TO-NDRL Subsystem.
N8	Receive subresponse-size from the EXTRACT-AND-COMPUTE-DATA Subsystem.
N9	Send subresponse-size to the IMPLEMENT-COMPOSITION-STRATEGY Subsystem.
QMC|D2Q	
N10	Receive response-composition-program the IMPLEMENT-COMPOSITION-STRATEGY Subsystem.
N11	Receive subresponse assembly-node-ids from the IMPLEMENT-COMPOSITION-STRATEGY Subsystem.
N12	Send subresponse assembly-node-ids to the PREPARE-OUTPUT-FOR-NETWORK Subsystem.
N13	Send perform-SEMI-JOIN-select message to the ASSEMBLE-COMPOSITE-RESPONSE Subsystem.
N14	Send perform-SEMI-JOIN-join message to the ASSEMBLE-COMPOSITE-RESPONSE Subsystem.
N15	Send perform-SEMI-JOIN-append message to the ASSEMBLE-COMPOSITE-RESPONSE Subsystem.

NORMAL	EVENT DESCRIPTION
N16	Send subresponse-status-data-request to the ASSEMBLE-COMPOSITE-RESPONSE Subsystem.
N17	Receive subresponse-status-data from the ASSEMBLE-COMPOSITE-RESPONSE Subsystem.
N18	Send send-output-directive to the ASSEMBLE-COMPOSITE-RESPONSE Subsystem.
N19	Receive response-composition-complete messages from the ASSEMBLE-COMPOSITE-RESPONSE Subsystem.
N20	Send Transmit-response-to-user message when all response-composition-complete messages have been received.

ERROR	EVENT DESCRIPTION
QMC|D2Q	
E1	Subresponse execution error. Abort subresponse.
E2	Subresponse over quota condition. Abort subresponse.
E3	Unable to transmit subresponse error. Retry or abort subresponse.
E4	Subresponse composition error. Abort Subresponse.
E5	Response over quota condition. Abort response.
E6	Precedence-override-condition. Halt query processing.

RESPONSE	EVENT DESCRIPTION
QMC|D2Q	
R1	QMC provides alternate address if subresponse transmission error has occurred.
R2	QMC cancels subresponse processing during precedence-override-condition.

ERROR	EVENT DESCRIPTION
R3	QMC restarts low-precedence-query processing.
R4	QMC purges query from IDN due to Administrative Override.

TERMINAL	EVENT DESCRIPTION
QMC|D2Q	
T1	Terminate due to subresponse execution error.
T2	Terminate due to subresponse over quota condition.
T3	Terminate due to unable-to-transmit subresponse errors.
T4	Terminate due to subresponse composition error.
T5	Terminate due to response over quota condition.
T6	Terminate due to administrator override condition.

3.7.3 TECHNICAL REQUIREMENTS

3.7.3.1 Actions

This specification of the technical requirements for QMC is designed to accompany the Data Flow Diagram and Event Table presented earlier. For clarity, the actions are grouped by processor type.

3.7.3.1.1 D1 QMC Processing

The QUERY-MONITORING-AND-CONTROL Subsystem first becomes aware of the submission of a query when the VALIDATE-TOKENIZED-QUERY Subsystem, at a D1 Node, has created a QCB for the new query and requested a D2Q Node destination-id for it. QMC at a D1 Node (QMC|D1):

- Consults the network status information provided by the Network Information (NI) Processors.

- Determines a D2Q to process the query (this becomes the Controlling D2Q, i.e., the D2Q/C Node).

- Passes the destination-id for the D2Q/C Node to the Validated Tokenized Query.

- Notifies QMC at the D2Q/C where final output is being held, and when it is released to User Node.

3.7.3.1.2 D2Q/C QMC Processing of the Query and Subqueries

QMC at the D2Q/C Node (QMC|D2Q/C) takes control of the Query Control Block (QCB) from the VALIDATE-TOKENIZED-QUERY Subsystem. QMC at the D2Q/C provides for the initial handling of the query, and it is also responsible for monitoring the progress of the query until the response is finally delivered to the user. The D2Q/C is, then, the "controlling node" for the particular query.

The following major processing steps occur within the D2Q/C Node:

- The QCB is passed to the VALIDATE-NORMALIZED-QUERY Subsystem by QMC|D1.

- If the query passes all of the validation tests, QMC|D2Q passes the QCB to the DETERMINE-DATA-ACCESSIBILITY Subsystem.

- Next, QMC|D2Q/C transfers the QCB to the DECOMPOSE-INTO-SUBQUERIES Subsystem for further processing. The result of processing by this Subsystem is a set of one or more Subquery Control Blocks (SQCBs). The SQCBs include the canonical form subqueries derived from the original query.

- QMC|D2Q/C takes control of the SQCBs and passes them to the IMPLEMENT-EXECUTION-STRATEGY Subsystem. The execution strategy, which is contained in the Query Execution Graph (QEG), is generated by this Subsystem and passed back to QMC|D2Q/C.

- For the Data Nodes to which the subqueries are targetted, QMC|D2Q/C next passes the associated SQCBs to the RECEIVE-SUBQUERIES Subsystem at the companion D1* Nodes .

3.7.3.1.3 D1* QMC Processing of the Subqueries

The subqueries can be queued for processing against the target databases at the Data Node. QMC at the D1* (QMC|D1*):

- Inspects the input queue for SQCBs.

- Passes each SQCB, on a precedence basis, to the TRANSLATE-TO-NDRL Subsystem which transforms the canonical form Subquery to the Network Data Request Language form.

- Receives the translated Subquery and transfers it to the EXTRACT-AND-COMPUTE-DATA Subsystem at the Data Node.

3.7.3.1.4 D1* and D2Q/C QMC Processing of Subresponses and Responses

If there are no execution errors, the following actions occur:

- The EXTRACT-AND-COMPUTE-DATA Subsystem sends a subresponse-size message to QMC|D1*.

- QMC|D1* forwards the size message to the IMPLEMENT-COMPOSITION-STRATEGY Subsystem at D2Q Node.

- The IMPLEMENT-COMPOSITION-STRATEGY Subsystem computes a response composition strategy using the SEMI-JOIN algorithm and generates a Response Composition Program.

- In parallel with this calculation, the EXTRACT-AND-COMPUTE-DATA Subsystem sends the subresponse from the Data Node to the PREPARE-OUTPUT-FOR-NETWORK Subsystem at the companion D1* Node.

- In order to determine the initial assembly point for response composition, the PREPARE-OUTPUT-FOR-NETWORK Subsystem sends an assembly-node-id request to QMC at D2Q/C Node. (N.B.: The composition sites are likely to be all the D1* Nodes at which the subresponses are received.)

- QMC at the D2Q/C has now received the Response Composition Program from the IMPLEMENT-COMPOSITION-STRATEGY Subsystem along with a set of assembly-node-ids for the subresponses.

- QMC sends an assembly-node-id to the PREPARE-OUTPUT-FOR-NETWORK Subsystem at each D1* Node which has a subresponse to the query.

- The subresponses are then transmitted to the initial assembly nodes (D1* Nodes) and are queued for the ASSEMBLE-COMPOSITE-RESPONSE Subsystem at those nodes.

QMC at the D2Q/C is now responsible for interpreting the Response Composition Program and sending a series of subresponse processing commands to the ASSEMBLE-COMPOSITE-RESPONSE Subsystems at the nodes where the subresponses currently reside. These commands, which are explained in the ASSEMBLE-COMPOSITE-RESPONSE Subsystem specification, are given in the following table:

COMMAND	FROM	TO
Perform-SEMI-JOIN-select	QMC|D2Q/C	ASSEMBLE-COMPOSITE-RESPONSE
Perform-SEMI-JOIN-join	QMC|D2Q/C	ASSEMBLE-COMPOSITE-RESPONSE
Perform-SEMI-JOIN-append	QMC|D2Q/C	ASSEMBLE-COMPOSITE-RESPONSE
Send-output directive	QMC|D2Q/C	ASSEMBLE-COMPOSITE-RESPONSE

After each command is sent, a subresponse-status-data-request is sent to the ASSEMBLE-COMPOSITE-RESPONSE Subsystems and a subresponse-status-data message is received by QMC at the D2Q/C. When all subresponses have been joined into a single composite response, a response-composition-complete message is sent to QMC at the D2Q/C and a transmit-response-to-user message is returned by QMC|D2Q to the node which is holding the response.

3.7.3.1.5 QMC Purge of the Query from the IDN

QMC|D1 waits until a response-delivered-to-user message is sent from the DELIVER-COMPOSITE-RESPONSE Subsystem at the D1 Node to the QMC Process at that same node. Upon receipt of this message by QMC|D1, the following steps occur:

- QMC determines the D2Q/C for the query from the destination-id in the QCB at the D1.
- A purge-QCB-SQCB-QEG message is sent to QMC at the D2Q/C for the particular query.
- QMC at the D2Q/C uses the QEG and the Response Composition Assembly node id list to determine the D1* and D2Q Nodes which participated in the processing of the query.

- QMC at the D2Q/C broadcasts a message to all D2Q and D1* Nodes participating in the processing of the query or its response, including any "shadow D2Q" (D2Q/S).

- QMC at each of these nodes outputs the appropriate purge message(s) to the local IRDS in order to clear the processed query from the IDN.

3.7.3.2 Intrasubsystem Data Flows

The major intrasubsystem data flows to be considered for QMC occur among the QMC Processes resident at each of the D1, D2Q, and D1* Node types. These messages include:

- IDN status messages.

- Precedence-override-condition messages.

- Query processing restart messages.

Figure 3.7.3 shows how these messages are passed among the components of the QUERY-MONITORING-AND-CONTROL Subsystem.

3.7.3.3 Tasking

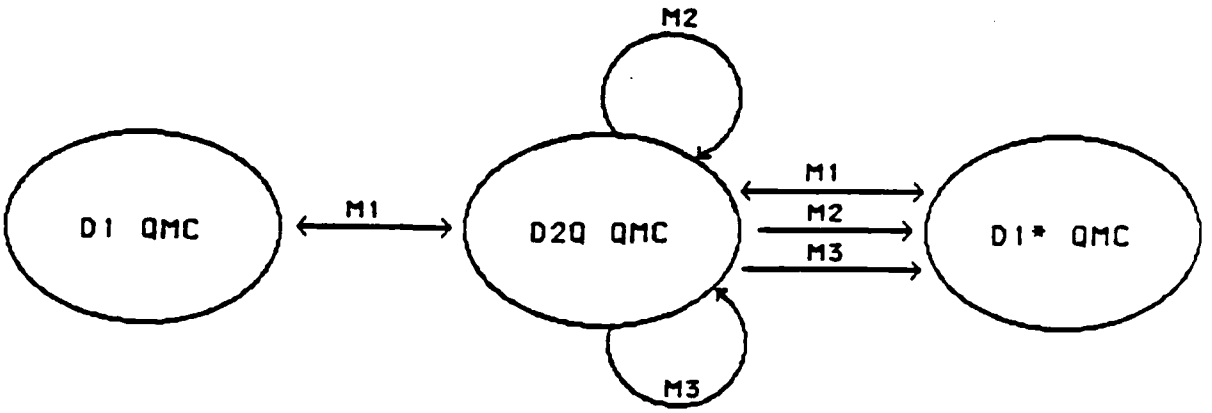
QMC can be viewed as a set of cooperating tasks. There is one QMC task at each of the D1, D2Q and D1* processors. The nature of of each task varies by the type of the associated processor.

3.7.3.4 Exception Conditions

The major exception conditions recognized by QMC include:

- Subresponse execution error.

Subsystem 7: QUERY-MONITORING-AND-CONTROL .



Message Type	Message Description
M1	IDN Status
M2	Precedence Override
M3	Query Processing Restart

Figure 3.7.3

- Subresponse over quota condition.
- Unable-to-transmit subresponse error.
- Subresponse composition error.
- Response over quota condition.
- Precedence override condition.

3.7.3.5 Rationale for Critical Decisions

The decision to divide QMC into coordinated processes at each D1, D2Q, and D1* Node was the most critical specification decision made for this Subsystem.

This approach results in a significant reduction of control and status messages over a pure central control (single node) approach. It also provides a robust character to the IDN in that QMC at any node may be used to restart the processing of a query if it is interrupted.

3.7.3.6 Open / Deferred Decisions

None.

3.7.3.7 Critical Algorithms

The critical algorithms for QMC are:

- The Subquery Execution Strategy.
- The Response Composition Strategy.

These algorithms are embodied in the IMPLEMENT-EXECUTION-STRATEGY Subsystem and IMPLEMENT-COMPOSITION-STRATEGY Subsystem, respectively.

3.7.4 DATA REQUIREMENTS

3.7.4.1 Major Data Flows

The major data flows into and out from QMC are listed below. Descriptions of each data flow can be found under the Subsystem Specifications for the particular Subsystems which send or receive them.

DATA ITEM	IN/OUT	SUBSYSTEM	PROCESSOR TYPE
destination-request	I	the VALIDATE-TOKENIZED-QUERY Subsystem	D1
destination-data	O	the VALIDATE-TOKENIZED-QUERY Subsystem	D1
normalized-query	I	the VALIDATE-TOKENIZED-QUERY Subsystem	D1
	O	the VALIDATE-INTERMEDIATE-QUERY Subsystem	D2Q/C
QCB	I	the VALIDATE-INTERMEDIATE-QUERY Subsystem	D2Q/C
	O/I	the DETERMINE-DATA-ACCESSIBILITY Subsystem	D2Q/C
	O/I	the DECOMPOSE-INTO-SUBQUERIES Subsystem	D2Q/C
SQCBs	I	the DECOMPOSE-INTO-SUBQUERIES Subsystem	D2Q/C
	O	the DEVELOP-EXECUTION-STRATEGY Subsystem	D2Q/C

DATA ITEM	IN/OUT	SUBSYSTEM	PROCESSOR TYPE
	O	the RECEIVE-SUBQUERIES-AT-DESTINATION Subsystem	D1*
	O	the TRANSLATE-TO-NDRL Subsystem	D1*
QEG	I	the DEVELOP-EXECUTION-STRATEGY Subsystem	D2Q/C
subquery-execution-error	I	the EXTRACT-AND-COMPUTE-DATA Subsystem	DATA NODE
subresponse-size	I	the EXTRACT-AND-COMPUTE-DATA Subsystem	DATA NODE
subresponse-over-quota	I	the EXTRACT-AND-COMPUTE-DATA Subsystem	DATA NODE
abort-subquery-processing	O	the EXTRACT-AND-COMPUTE-DATA Subsystem	DATA NODE
assembly-node-id-request	I	the PREPARE-OUTPUT-FOR-NETWORK Subsystem	D1*
assembly-node-id	O	the PREPARE-OUTPUT-FOR-NETWORK Subsystem	D1*
subresponse-size	I	the DEVELOP-COMPOSITION-STRATEGY Subsystem	D2Q/C
response-composition-program	O	the DEVELOP-COMPOSITION-STRATEGY Subsystem	D2Q/C
subresponse-status-data	I	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem	D1*+D2Q
output-transmission-error	I	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem	D1*+D2Q
response-composition-error	I	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem	D1*+D2Q
response-composition-complete	I	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem	D1*+D2Q
perform-SEMI-JOIN-select	O	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem	D1*+D2Q

DATA ITEM	IN/OUT	SUBSYSTEM	PROCESSOR TYPE
perform-SEMI-JOIN-join	O	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem	D1*+D2Q
perform-SEMI-JOIN-append	O	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem	D1*+D2Q
send-output-directive	O	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem	D1*+D2Q
cancel-output-directive	O	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem	D1*+D2Q
subresponse-status-request	O	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem	D1*+D2Q
transmit-response-to-user	O	the ASSEMBLE-COMPOSITE-RESPONSE Subsystem	D1*+D2Q
response-delivered-to user	I	the DELIVER-COMPOSITE-RESPONSE Subsystem	D1
purge-QCB-SQCBs-QEG	O/I	the QUERY-MONITORING-AND-CONTROL Subsystem|D1 to the QUERY-MONITORING-AND-CONTROL Subsystem|D2Q/D1*	D1
purge-SQCBs	O/I	the QUERY-MONITORING-AND-CONTROL Subsystem|D1 to the QUERY-MONITORING-AND-CONTROL Subsystem|D1*	D1*
purge-QCB-SQCBs	O/I	the QUERY-MONITORING-AND CONTROL Subsystem|D1 to the QUERY-MONITORING-AND-CONTROL Subsystem|D2Q	D2Q/C/S
purge-QEG	O/I	the QUERY-MONITORING-AND-CONTROL Subsystem|D2Q to the QUERY-MONITORING-AND-CONTROL Subsystem|D2Q	D2Q/C/S

DATA ITEM	IN/OUT	SUBSYSTEM	PROCESSOR TYPE
purge-QCB	O/I	the QUERY-MONITORING-AND CONTROL Subsystem|D1 to the QUERY-MONITORING-AND-CONTROL Subsystem|D1	D1
restart-query-directive	O/I	the QUERY-MONITORING-AND CONTROL Subsystem|D1 to the QUERY-MONITORING-AND-CONTROL Subsystem|D1/D2Q/D1*	D1+D2Q+D1*

3.7.4.2 Special Data Requirements

QMC is dependent upon network status information provided by the Network Information (NI) processors.

3.7.5 DATABASE REQUIREMENTS

3.7.5.1 Network Metadata

The QUERY-MONITORING-AND-CONTROL Subsystem uses both static and dynamic network metadata to determine node availability and to assign destination addresses.

3.7.5.2 Data Metadata

The QUERY-MONITORING-AND-CONTROL Subsystem consults the Query Execution Graph (QEG) to determine if a subresponse must be used in as a subsequent Subquery.

QMC executes the Response Composition Program in order to join subresponses into a composite response.

3.7.5.3 Other

None.

3.7.6 NOTES

None.

3.8 SUBSYSTEM 8: RECEIVE-SUBQUERIES

3.8.1 OVERVIEW

Subsystem 8, the RECEIVE-SUBQUERIES Subsystem, is designed to provide a link between the processing performed at the controlling D2Q Node (D2Q/C) and the TRANSLATE-TO-NDRL Subsystem (Subsystem 9) at a D1* Node.

This Subsystem receives the canonical form Subquery Control Blocks (SQCBs) from the QUERY-MONITORING-AND-CONTROL (QMC) Subsystem, (Subsystem 7), after they have been processed by the IMPLEMENT-SUBQUERY-PROCESSING-STRATEGY Subsystem (Subsystem 6). It stores the SQCBs in the Translation-Input-Queue for subsequent processing by Subsystem 9.

Subsystem 8 relies upon the underlying protocol to properly transmit the SQCBs from the D2Q/C Node to the D1* Node. If an error, such as the failure to receive a SQCB or the receipt of only a partial SQCB, occurs, QMC is notified and an attempt is made to retransmit the SQCB by QMC. If an error is detected within the D1* Node itself, QMC must determine an alternate destination for the subquery, if one is available, or the subquery must be terminated.

3.8.2 INTERFACE SPECIFICATIONS

3.8.2.1 Intersubsystem Data Flows

Figure 3.8.1 shows the data flows between this and other Subsystems.

Subsystem 8: RECEIVE-SUBQUERIES

In	Data Item	Source Subsystem
I1	SQCB	QUERY-MONITORING-AND-CONTROL
I2	Verify-receipt-of-SQCB-msg	QUERY-MONITORING-AND-CONTROL

Out	Data Item	Destination Subsystem
O1	SQCB	Translate to NDRL
O2	SQCB-received-msg	QUERY-MONITORING-AND-CONTROL
O3	Input-queue-full-msg	QUERY-MONITORING-AND-CONTROL

Figure 3.8.1

3.8.2.2 Subsystem Events

The event diagram for this Subsystem is shown in Figure 3.8.2.

3.8.3 TECHNICAL REQUIREMENTS

3.8.3.1 Actions

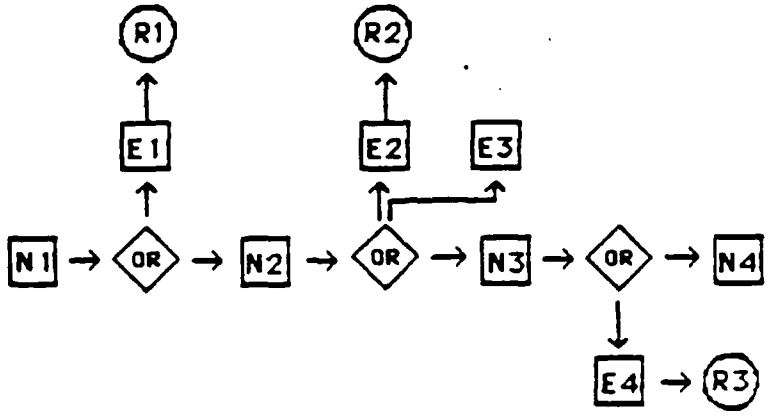
The RECEIVE-SUBQUERIES Subsystem is responsible for receiving subqueries, performing simple validation and completeness checks, and then depositing valid SQCBs in the Translation-Input-Queue for processing by the TRANSLATE-TO-NDRL Subsystem (Subsystem 9). This Subsystem is under the supervision of QMC.

The RECEIVE-SUBQUERIES Subsystem is entirely data driven, therefore, many actions may occur asynchronously. The general sequence of events is as follows:

- QMC|D2Q transmits an SQCB to the destination D1* Node.
- The RECEIVE-SUBQUERIES Subsystem accepts the SQCB on behalf of the D1* Node.
- The SQCB is checked for completeness.
- The validated SQCB is placed in the Translation-Input-Queue.
- A message is sent to QMC|D2Q and QMC|D1* confirming the receipt of a valid SQCB.

The SQCB is validated by comparing it to the control information stored in the D1* dictionary. If a component of the SQCB is missing,

Subsystem 8: RECEIVE-SUBQUERIES



Normal	Event Description
N1	Receive the SQCB from the D2Q/C
N2	Check SQCB for completeness
N3	Place SQCB in the Translation-Input-Queue
N4	Notify QMCID1* and QMCID2Q of receipt of valid SQCB

Error	Event Description
E1	SQCB not yet received
E2	SQCB incomplete
E3	SQCB invalid
E4	Translation-Input-Queue is full

Response	Event Description
R1	SQCB received / not received
R2	SQCB invalid or incomplete
R3	Translation-Input-Queue full

Figure 3.8.2

then the SQCB is judged incomplete. If there is no data within one or more required components the SQCB is judged invalid.

If QMC has not received confirmation of the processing of the SQCB from the D1* Node within some designated timeout period, then a message is sent to the RECEIVE-SUBQUERIES Subsystem at this D1* Node to determine if the D1* Node is available and to notify the Subsystem that an SQCB should have been received. The RECEIVE-SUBQUERIES Subsystem can, in this case, respond to QMC in one of three ways:

- o No response - the D1* Node is out of service.
- o SQCB received - subquery processing will continue.
- o Valid SQCB not received - a new SQCB copy should be sent.

Since all SQCBs have a unique identifier (i.e., unique over an arbitrarily long time period of time), the RECEIVE-SUBQUERIES Subsystem is able to discard any redundant copies of an SQCB that it might receive due to delays in transmission or processing.

In addition to the responses cited above, the RECEIVE-SUBQUERIES Subsystem can also generate error responses. These are discussed in Section 3.8.3.4, Exception Conditions.

3.8.3.2 Intrasubsystem Data Flows

Since this Subsystem runs as a single task, there are no intrasubsystem data flows. SQCBs are received from the D2Q/C Node, validated, and deposited in the Translation-Input-Queue for further processing. This flow is presented in the Intersubsystem Data Flow Diagram in Section 3.8.2.1.

3.8.3.3 Tasking

This Subsystem is composed of a single task for each subquery.

3.8.3.4 Exception Conditions

The RECEIVE-SUBQUERIES Subsystem can respond to three error conditions:

- Subquery invalid.
- Subquery incomplete.
- Translation-Input-Queue full.

If any of these conditions occur, an appropriate response is sent to QMC. QMC must note the type of error that occurs in order to determine if the SQCB should be:

- Retransmitted to the same D1* Node.
- Transmitted to a different D1* Node if one is available.
- Held in the SQCB-Output-Queue at the D2Q/C Node until an appropriate D1* Node is available.
- Cancelled and the cancellation noted in the QEG.

If QMC receives no response (over a timeout period) from the D1* Node, even after an SQCB verification message has been sent, the QMC must assume that the D1* Node is out of service, and either the SQCB must be sent to another D1* Node (if an appropriate destination is available), or it must be cancelled.

3.8.3.5 Rationale for Critical Decisions

1. The critical decision for this Subsystem was whether it should be designed as a separate Subsystem or should be bundled in with Subsystem 9, the TRANSLATE-TO-NDRL Subsystem.

   Since the receipt of SQCBs at the D1* is both logically and physically a separate task from that of Subsystem 9, it is preferrable to maintain it as a separate function. The RECEIVE-SUBQUERIES Subsystem can, however, be viewed as a component of the QUERY-MONITORING-AND-CONTROL Subsystem (QMC) at the D1* Node, given its close relationship with that Subsystem.

3.8.3.6 Open / Deferred Decisions

None.

3.8.3.7 Critical Algorithms

None.

3.8.4 DATA REQUIREMENTS

3.8.4.1 Major Data Flows

Canonical SQCB:

The Subquery Control Block as transmitted from Subsystem 6 at the D2Q/C Node to the D1* Node.

3.8.4.2 Special Data Requirements

The schema of the Dl* dictionary provides a "template" of the SQCB structure and format which is used by Subsystem 8 for SQCB validation. See Figure 3.8.3 (Dl* IRDS View).

3.8.5 DATABASE REQUIREMENTS

3.8.5.1 Network Metadata

The QUERY-MONITORING-AND-CONTROL Subsystem must use both static and dynamic network metadata to determine appropriate nodes and assign destination addresses if an alternate Dl* has to be found for a SQCB.

3.8.5.2 Data Metadata

The QUERY-MONITORING-AND-CONTROL Subsystem must consult the Query Execution Graph to determine where every SQCB has been sent.

3.8.5.3 Other

None.

3.8.6 NOTES

None.

D1* IRDS VIEW

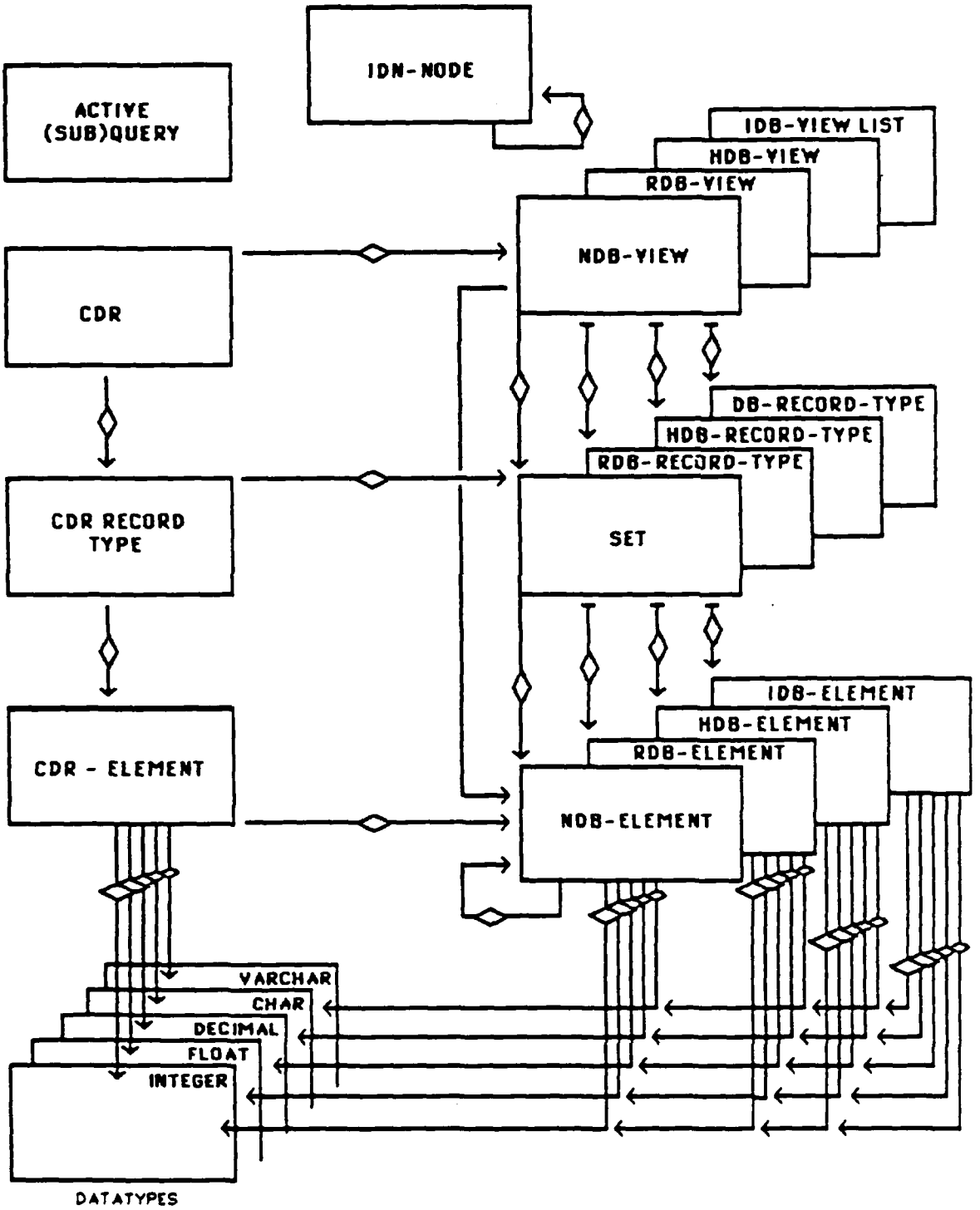


Figure 3.8.3

(This page intentionally left blank)

3.9 SUBSYSTEM 9: TRANSLATE-TO-NETWORK-DATA-REQUEST-LANGUAGE

3.9.1 OVERVIEW

Subsystem 9, the TRANSLATE-TO-NETWORK-DATA-REQUEST-LANGUAGE Subsystem, is executed at D1* Nodes. This Subsystem is responsible for converting the Subqueries that it receives (from Subsystem 8, the RECEIVE-SUBQUERIES Subsystem) in IDN canonical form to the NETWORK DATA REQUEST LANGUAGE (NDRL) syntax. The subquery, expressed in the NDRL, is then ready for transmission to the EXTRACT-AND-COMPUTE-DATA Subsystem, at a Data Node and Subsystem 10, where the final phase of translation translation into native DBMS format takes place. A detailed overview of the query / subquery translation process is provided in the document: Subquery Processing Against the Basic Data Models.

Although the NDRL cannot be processed directly against a target database, this translation is necessary. Translation to NDRL insures that the subquery is represented in a form which is semantically correct and complete and which can be mapped syntactically to the native data query or report writer language of the target DBMS. The amount of additional processing required at the Data Node to convert the subquery to native format is thus minimized.

The NDRL is designed to be a "high level", "English-like", language which is consistent with the languages currently used to access most relational databases and is similar to SQL (Structured Query Language). Patterning the NDRL after SQL provides the available universal medium for subqueries which are to be executed against heterogeneous databases. Once the subquery has been translated into this form, it can be passed to the EXTRACT-AND-COMPUTE-DATA Subsystem (Subsystem 10) at the Data

Node for conversion to native DBMS syntax and processing against databases which use any of the four standard data models: hierarchical, network, relational, or inverted file.

The process of translation to NDRL relies upon the database subschemas which are stored in the dictionary at the D1* Node. Just as the CDRs at the D2Q Node provide the information necessary for decomposition of the query into subqueries, the database subschemas provide the detailed information necessary for translation of the canonical form subquery to the NDRL. The database subschema provides a detailed "logical view" of the target database in terms of the target database's data model. This information is requested through the IRDS at the D1* Node.

In addition to the subschema for the target database, the D1* dictionary also provides the Database Access Names (DANS) which correspond to the Network Access Names (NANS) used for the data elements within the subquery. The NANS are unique across the entire IDN while the DANS are specific to the target database and may not be unique (i.e., they may be qualified by a record name).

3.9.2 INTERFACE SPECIFICATIONS

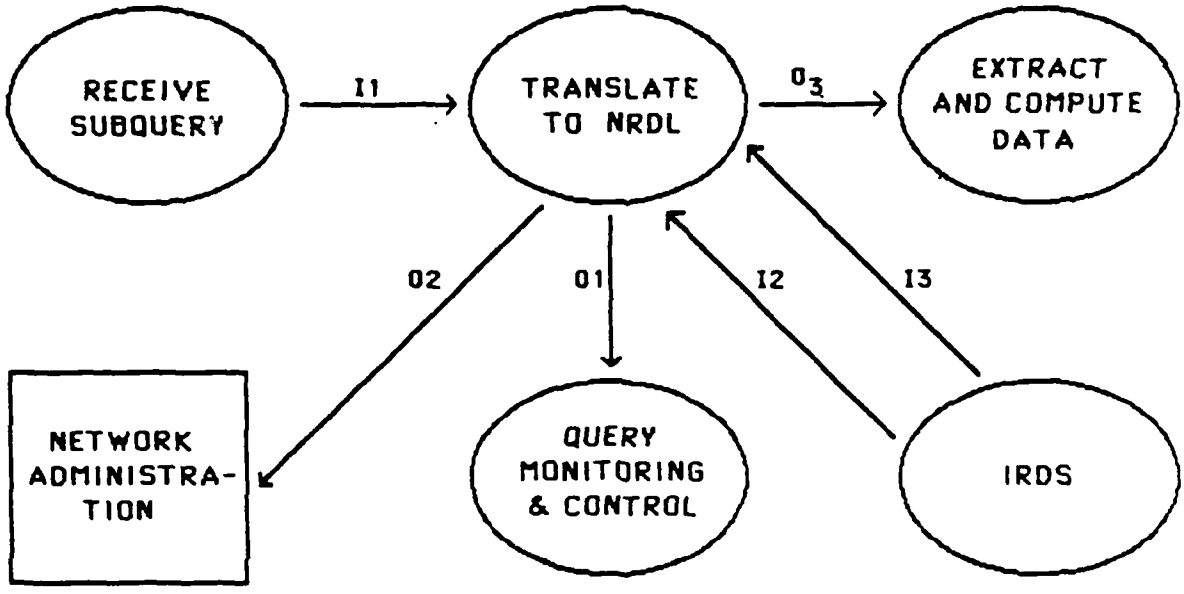
3.9.2.1 Intersubsystem Data Flows

Figure 3.9.1 shows the data flows between this and other Subsystems.

3.9.2.2 Subsystem Events

The event diagram for this Subsystem is shown in Figure 3.9.2.

Subsystem 9: TRANSLATE-TO-NDRL

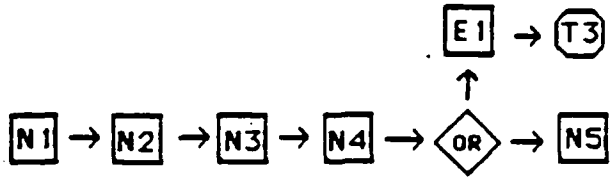


In	Data Item	Source Subsystem
I1	SQCB	RECEIVE-SUBQUERIES
I2	DANS-identifiers	IRDS
I3	Database-subschema	IRDS

Out	Data Item	Destination Subsystem
O1	Subquery-translation-error	QUERY-MONITORING-AND-CONTROL
O2	Subquery-translation-error	Network Administration
O3	NRDL-Subquery	EXTRACT-AND-COMPUTE-DATA

Figure 3.9.1

Subsystem 9: TRANSLATE-TO-NDRL



Normal	Event Description
N1	Input canonical form subquery from D1* processing queue (Translation-Input-Queue)
N2	Lookup the DANs identifiers for the subquery data elements
N3	Receive subschema for target database from local IRDS
N4	Translate canonical subquery to NDRL subquery
N5	Transmit NDRL subquery to the target Data Node

Error	Event Description
E1	Error encountered in the translation of subquery to NDRL

Terminal	Event Description
T1	Termination due to translation error

Figure 3.9.2

3.9.3 TECHNICAL REQUIREMENTS

3.9.3.1 Actions

The Subquery Control Block (SQCB) is input from the processing queue at the D1* Node. The Subquery was placed in the queue by the previous Subsystem, the RECEIVE-SUBQUERIES-AT-DESTINATION Subsystem. The TRANSLATE-TO-NDRL Subsystem relies upon information stored in the node IRD to convert the subquery from canonical form to the Network Data Request Language Once the translation process is complete, the NDRL Subquery is queued for transmission to the target Data Node.

The translation process itself proceeds, in general, as follows:

- The identifier for the target database for the subquery is determined from the SQCB.
- The NANS for each data element within the subquery is used as a key to look up the corresponding DANS for the data elements in the target database and replace the NANS with DANS in the NDRL subquery.
- The identifiers for each logical record in the target database from which data element values are to be extracted are determined from the subschema for the target database and included in the NDRL subquery. Note that the concept of "logical record" is different for each of the standard data models and these differences are reflected in the database subschemas.
- Any other required structural information (such as logical view names, area names, set names) needed for the processing of the subquery against the target database is obtained from the subschema and included in the NDRL subquery.

- If no errors have been encountered in the subquery translation process, the completed NDRL Subquery is queued for transmission to the Data Node for translation to native DBMS format and execution against the target database.

Further details of the NDRL translation process are given in Section 3.9.3.7, Critical Algorithms. A complete description of the entire subquery translation process, through all of its phases, is given in the paper: Subquery Processing Against the Basic Data Models. The NDRL itself is specified as follows:

- SELECT clause:

  SELECT dans-1, dans-2, ...

  where dans-i are the uniquely specified Database Access Names for the data elements to be queried in the target database;

  dans-i must be unique to the database and may take one of two forms:

  simple-database-access-name

  qualified-database-access-name

  qualified-database-access-name is of the form:

  record-name.simple-database-access-name

  record-name is the name of the logical record within which the data element resides.

- FROM clause:

  FROM record-name-1, record-name-2, ...

where record-name is the unique record identifier for the record in which a selected data element resides.

[N.B.: It is assumed that all record names in the subschema and in the corresponding target database will be unique. If this is not the case for a particular database, then the record names for that database will have to be referenced in the subschema in fully qualified form using the "dot" notation (e.g., view-name.record-name).]

o WHERE clause:

WHERE dans-i <relational-operator> ({value-j | dans-k})

(<logical-operator>) ...

[N.B.: A WHERE clause with at least one restriction subclause must appear within the NDRL query since unrestricted subqueries are disallowed. The required restriction, however, may be of the form: dans-i EXISTS.]

where:

dans-i is the Database Access Name for a data element name in the SELECT clause.

[N.B.: restrictions are not permitted on a data element name which does not appear in the SELECT clause since this would be inconsistent with the original specification format for the query.]

relational-operator may be any one of the following:

EQ	-	equals
GT	-	greater than

LT	-	less than
LE	-	less than or equal to
GT	-	greater than or equal to
NE	-	not equal to
EXISTS	-	a non-null value exists
FAILS	-	the value is null
CT	-	contains

[N.B.: Although the EXISTS and FAILS operators are generic, the particular representation of a null value within the database is dependent upon the individual DBMS. This must be taken into account when the subquery is translated into native format.]

value-j may be any one of the following:

an integer
a decimal quantity
a character string, enclosed in double quotes ("...")

character strings may contain:

ASCII characters
numerals expressed as ASCII characters
blanks
any special characters other than a double quote (")
? as a wildcard character
* as a wildcard character

[N.B.: Since wildcard characters are not supported by some DBMS query facilities, restrictions which include wildcards may have to be converted to prefix, suffix, or even EXISTS restrictions when the subquery is translated to native format.]

dans-j is the Database Access Name for any data element specified in the SELECT clause.

logical-operator is any one of the following:

AND
OR
NOT

[N.B.: Other logical operators, such as XOR (exclusive OR) are not permitted since they are not commonly supported by DBMS query facilities.]

o OUTPUT clause:

OUTPUT <output-subclauses>

[N.B.: Output subclauses are required in order to insure that the subresponse which results from the subquery has been put in the proper format by the local DBMS.]

where <output-subclauses> are all of the following:

ORDERED-BY dans-i ...

HEADINGS OFF

FOOTINGS OFF

COLUMN SEPARATOR OFF

dans-i is the Database Access Name identifier of a data element which appears in the SELECT clause.

3.9.3.2 Intrasubsystem Data Flows

Figure 3.9.3 shows the intrasubsystem data flows among the functions and data stores of Subsystem 9.

3.9.3.3 Tasking

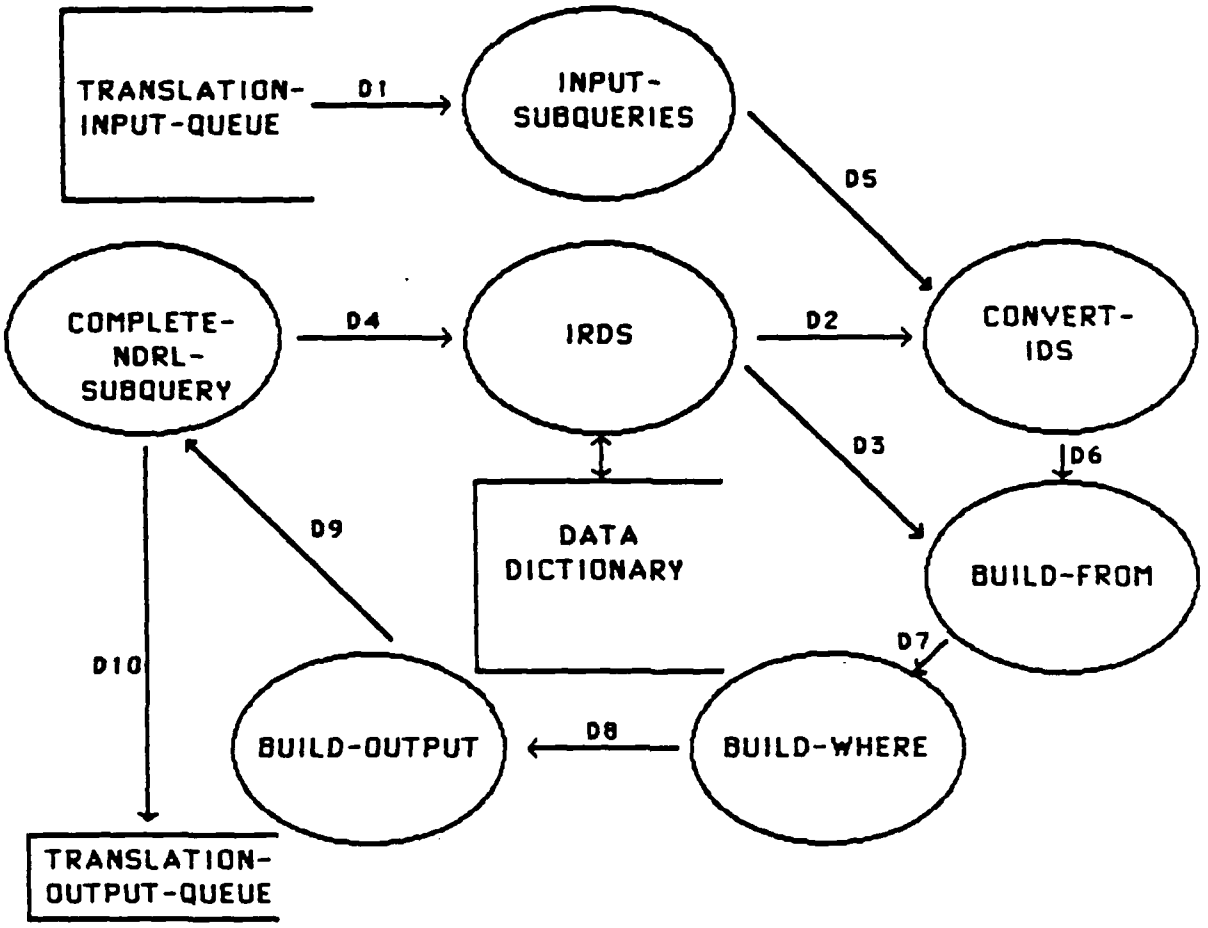
The TRANSLATE-TO-NDRL Subsystem may be viewed as a single task which is "data flow driven". The Translation-Input-Queue is continually checked to determine if any SQCBs are in the queue. If such a SQCB is found it is translated into an NDRL Subquery and placed in the Translation-Output-Queue for transmission to the Data Node.

3.9.3.4 Exception Conditions

A number of exceptions are possible during the TRANSLATE-TO-NDRL Subsystem processing. If the SQCB is properly formatted and the database subschemas are up to date, no translation error should occur. Any error which does occur must be considered severe since it not only terminates processing of the subquery but also indicates a problem within the IDN (e.g., an inconsistency between CDRs and subschemas). Such an error must be reported both to the QUERY-MONITORING-AND-CONTROL Subsystem (QMC) and to the Network Administrator. The following types of errors are possible:

- Unable to locate subschema for the database identified in the canonical SQCB.
- Unable to translate NANS to DANS.
- Unable to associate DANS with a record-name.
- Invalid data element restriction.

Subsystem 9: TRANSLATE-TO-NDRL



Data Flow	Data Item	Source Function / Store	Destination Function / Store
D1	SQCB	Translation-Input-Queue	Input-Subquery
D2	DANS-ids	IRDS	Convert-ids
D3	Record-names	IRDS	Build-FROM
D4	Access-names	IRDS	Complete-NDRL-Subquery
D5	canonical-subquery	Input-Subquery	Convert-ids
D6	identified-subquery	Convert-ids	Build-FROM
D7	qualified-subquery	Build-FROM	Build-WHERE
D8	restricted-subquery	Build-WHERE	Build-OUTPUT
D9	complete-subquery	Build-OUTPUT	Complete-NDRL-Subquery
D10	complete-NDRL-subquery	Complete-NDRL-SQCB	Translation-Output-Queue

Figure 3.9.3

- Invalid subquery specification within the canonical SQCB.
- Invalid SQCB.

If the Data Node is not available over a specified period of time then the subquery may also be terminated by QMC.

It is important to note that a terminated subquery does not terminate the entire query. QMC will continue to process the query as dictated by the Query Execution Graph.

3.9.3.5 Rationale for Critical Decisions

1. The critical decision for this Subsystem was concerned with the desirability of translation to NDRL before passing the subquery to the Data Node. The alternatives considered included:

   - Passing the canonical form subquery directly from the Dl* Node to the Data Node for translation into native format.
   - Translating the canonical form subquery into native format at the Dl* Node.

   The first alternative would place a much greater burden upon the Data Node than occurs by translating the subquery into the intermediate form of the NDRL at the Dl* Node. The overhead at the Data Node is greatly reduced and maintenance of the IDN software is facilitated since most of the software and metadata resides at the Dl*.

   The second alternative would require the Dl* Node to "know" the detailed characteristics and syntax of the query or report writer facility for each DBMS which could be accessed. Since a given Dl* Node might support several Data Nodes with several different DBMSs, the complexity of the translation at the Dl* Node would be greatly

increased. By first translating the subquery into NDRL at the D1* Node and then into native format at the Data Node, the translation process is simplified. In addition, maintenance of translation requirements particular to a given DBMS is isolated to the Data Node which employs that DBMS.

3.9.3.6 Open / Deferred Decisions

None.

3.9.3.7 Critical Algorithms

The critical algorithm for this Subsystem is the translation process itself. This process is described above under Section 3.9.3.1, Actions, and also in the paper: Subquery Processing Against the Basic Data Models.

The Translate-to-NDRL algorithm is basically a stepwise process of constructing the NDRL Subquery from the canonical SQCB. The major steps are:

- Translate the NANS to unique DANS, with any necessary qualification using record names, and build the SELECT clause.
- Build the FROM clause using the record names from the database subschema.
- Build the WHERE clause using the data element value restrictions specified in the canonical SQCB.
- Build the OUTPUT clause using any sort (ORDERED-BY) specifications in the SQCB.
- Complete the NDRL Subquery by including any additional access information required for the processing of the subquery. This

information (logical view names, area names, set names) is derived from the database subschema.

3.9.4 DATA REQUIREMENTS

3.9.4.1 Major Data Flows

SQCB

The Subquery Control Block which includes the canonical form subquery is constructed at the controlling D2Q Node and is passed to the D1* Node.

NDRL-Subquery

The NDRL subquery is constructed at the D1* Node and is passed to the Data Node.

3.9.4.2 Special Data Requirements

Translation-Input-Queue

A queue within the D1* Node which receives SQCBs from the RECEIVE-SUBQUERIES Subsystem and holds them for processing by the TRANSLATE-TO-NDRL Subsystem.

Translation-Output-Queue

A queue within the D1* Node which receives NDRL Subquery from the TRANSLATE-TO-NDRL Subsystem and holds them until they are transmitted to the target Data Node.

3.9.5. DATABASE REQUIREMENTS

3.9.5.1 Network Metadata

The QUERY-MONITORING-AND-CONTROL Subsystem must use the available dynamic network metadata to determine the status of the D1* Node and its companion Data Nodes.

3.9.5.2 Data Metadata

The TRANSLATE-TO-NDRL Subsystem must access the IRD at the D1* Node through the local IRDS in order to obtain:

- NANS to DANS translations.
- Record names associated which each data element.
- Access names needed for the execution of the subquery:
  - View names.
  - Area names.
  - Set names.

N.B.: any physical access information is supplied at the Data Node when the subquery is finally translated into native format. This information can include:

- Directory path names.

- Volume identifiers.
- Device logical names.

3.9.5.3 Other

None.

3.9.6 NOTES

All notes have been embedded within the sections to which they apply. There are no general notes for this Subsystem.

3.10 SUBSYSTEM 10: EXTRACT-AND-COMPUTE-DATA

3.10.1 OVERVIEW

Subsystem 10, the EXTRACT-AND-COMPUTE-DATA Subsystem, which executes at a Data Node, is responsible for the execution of subqueries, in Network Data Request Language (NDRL) format, against the target database at the Data Node. The interface between this Subsystem and the DBMS is through its native Data Manipulation Language (DML), Data Query Language (DQL) or Report Writer (RW) Facility. IDN software, which has been installed at the Data Node, is responsible for receiving subqueries from the D1* Interface Processor, performing whatever additional formatting is necessary for compatibility with the target database, and submitting the subqueries to the local DBMS.

The DBMS returns a subresponse for each subquery to this Subsystem, which, in turn, passes it to the D1* Node and the next Subsystem (the PREPARE-OUTPUT-FOR-NETWORK Subsystem). The output is in the form of a relational table with a virtual key, i.e., a code indicating the source database of the data. Control of the subresponse is then passed to the QUERY-MONITORING-AND-CONTROL Subsystem at the D1* Node for response assembly.

The process of subquery execution relies upon the fact that the subquery has been reduced to a set of simple selection criteria in NDRL format by the earlier Subsystems. The subresponse is produced in relational table format for the convenience of the ASSEMBLE-COMPOSITE-RESPONSE-Subsystem. Subsystem 10 also handles a variety of error conditions which may arise in the execution of subqueries.

3.10.2 INTERFACE SPECIFICATIONS

3.10.2.1 Intersubsystem Data Flows

Figure 3.10.1 shows the data flows between this and other Subsystems.

3.10.2.2 Subsystem Events

The event diagram for this Subsystem is shown in Figure 3.10.2.

3.10.3 TECHNICAL REQUIREMENTS

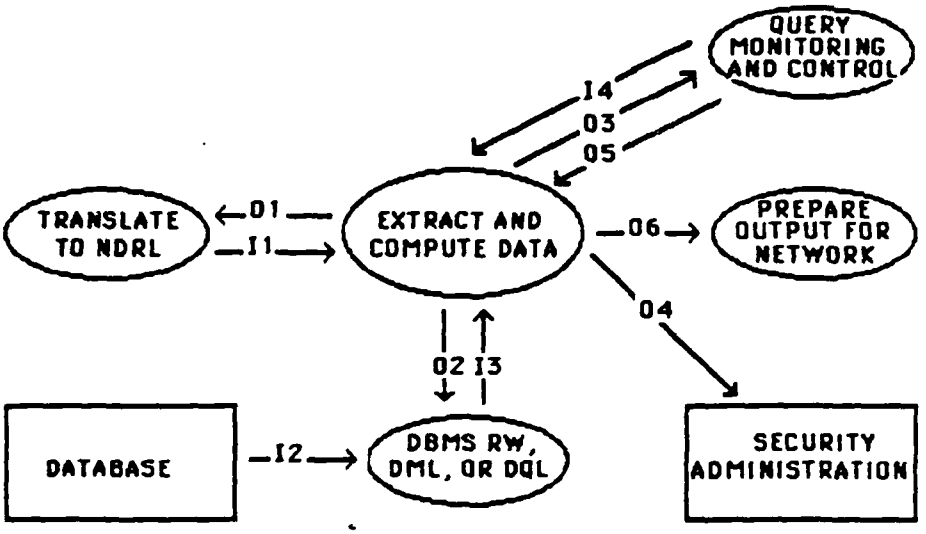
3.10.3.1 Actions

The subquery in Network Data Request Language (NDRL) format is passed by the previous Subsystem (TRANSLATE-TO-NDRL Subsystem) from the D1* Node to the Data Node. The database subschema which resides at the D1* was used by the previous Subsystem to make the subquery compatible with the target database.

Upon receipt of the subquery at the Data Node, the EXTRACT-AND-COMPUTE-DATA Subsystem must perform any additional translation that is required in order to present the subquery to the database. The subquery will be transformed into the native format of the local DBMS's DML, DQL, or Report Writer, whichever is appropriate. This translation will add such additional information to the subquery as:

- Physical database location (path, volume, area, etc.).
- Appropriate command keywords (e.g., FIND, SEARCH, etc.).

Subsystem 10: EXTRACT-AND-COMPUTE-DATA

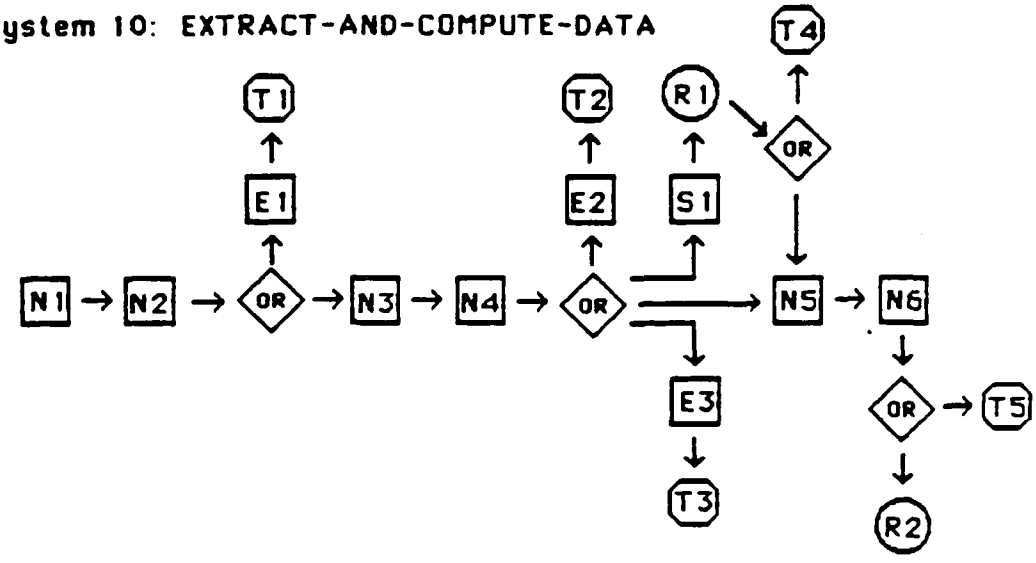


In	Data Item	Source Subsystem
I1	NDRL-subquery	TRANSLATE-TO-NDRL
I2	Local-Database-Data	Database
I3	Native-Format-Subresponse	Local DBMS
I4	Abort/Halt/Cont-Processing-Msg	QUERY-MONITORING-AND-CONTROL

Out	Data Item	Destination Subsystem
O1	Subquery-Execution-Error	TRANSLATE-TO-NDRL
O2	Native-Format-Subquery	Local DBMS
O3	Subresponse-Size-Message	QUERY-MONITORING-AND-CONTROL
O4	Security-Violation-Message	Security Administration
O5	Subresponse-Over-Quota-Message	QUERY-MONITORING-AND-CONTROL
O6	Native-Format-Subresponse	PREPARE-OUTPUT-FOR-NETWORK

Figure 3.10.1

Subsystem 10: EXTRACT-AND-COMPUTE-DATA



Normal	Event Description
N1	Receive NDRL subquery from D1*
N2	Translate subquery into native format
N3	Submit subquery to local DBMS
N4	Receive subresponse from local DBMS
N5	Construct virtual key and append to output records
N6	Send formatted subresponse on to next subsystem (at D1*)

Error	Event Description
E1	Error encountered in translation of subquery
E2	Error encountered in submitting subquery
E3	Subresponse exceeds absolute system quota

Special	Event Description
S1	Subresponse exceeds user quota

Response	Event Description
R1	User indicate abort/continue
R2	QUERY-MONITOTING-AND-CONTROL indicates abort

Terminal	Event Description
T1	Termination due to translation error
T2	Termination due to database error
T3	Termination due to exceeding absolute quota
T4	Termination due to User abort
T5	Termination by QUERY-MONITORING-AND-CONTROL

Figure 3.10.2

- Appropriate clause and subclause keywords (e.g., OWNER, etc.).
- Appropriate option specifications (e.g., NO DUPLICATES, etc.).
- Appropriate output formatting specifications:
  - Null Title and Column Headings.
  - Standard column separator (e.g., |).
  - Null footings and pagination.

If the subquery requires that computations be performed and the local DBMS is capable of performing them, then the subquery will include the required syntax for producing the calculated results. If the DBMS cannot perform the calculations, or, the user requests both detailed and summarized (computed) information, then the subquery will be formatted to generate the required records and the calculations will be performed at the D1* Node in the former case (PREPARE-OUTPUT-FOR-NETWORK Subsystem) and at the D1 Node in the latter case (DELIVER-COMPOSITE-RESPONSE Subsystem).

The subquery, in native format, is now submitted to the local DBMS and processed against the target database. The DBMS will return either the desired response (placing it in a designated file or buffer), or, error messages which may have been generated due to data value errors in the subquery. An explanation of error processing is given in Section 3.10.3.4.

The EXTRACT-AND-COMPUTE-DATA Subsystem will append a "virtual key" to the output; this key will permit the output to be readily distinguished from output generated from any other database in the IDN. The virtual key will be unique for databases with these different structures and semantics. It will be used in the subresponse composition processing. The virtual key is an internal data source identifier and it does not appear in the final output.

The subresponse generated by the local DBMS will be in relational table format. The output will consist of rows (records or tuples) and columns of data with a standard separator character between each column of data. There will be no extraneous information (such as headings and footings) permitted in the output. If it is not possible for the local DBMS to generate output in exactly this format, any extraneous information will be removed by the current Subsystem at the Data Node and subresponse formatting will be completed by the PREPARE-OUTPUT-FOR-NETWORK Subsystem at the D1* Node. In either case, the output, with the virtual key appended to each record, is now transmitted from the Data Node to the D1* Node.

3.10.3.2 Intrasubsystem Data Flows

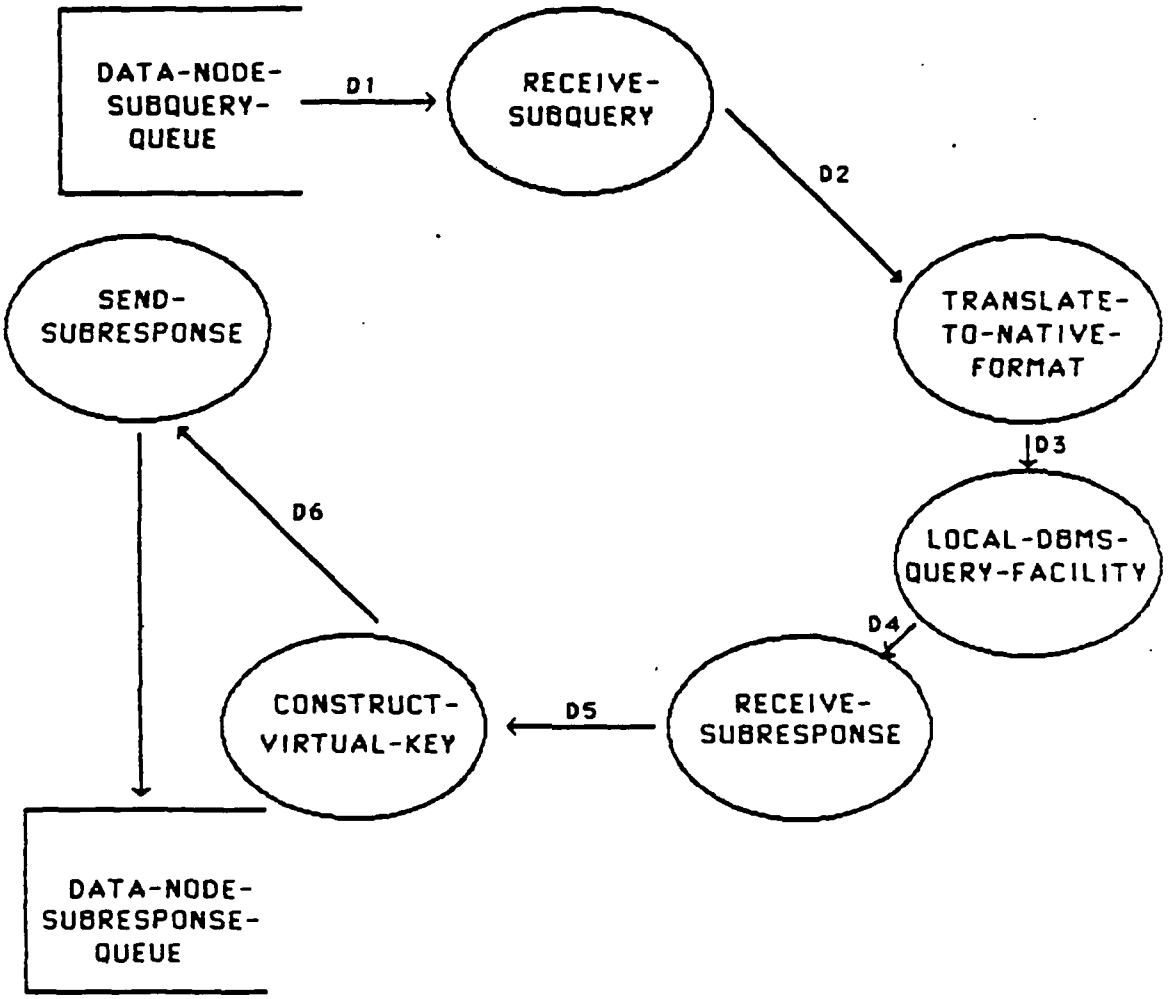
The intrasubsystem data flow diagram for the EXTRACT-AND-COMPUTE-DATA Subsystem is given in Figure 3.10.3.

These are the "normal" or mainstream internal data flows. A variety of additional data flows may occur due to error conditions, special conditions, and external responses. These are reviewed in Section 3.10.3.4.

3.10.3.3 Tasking

It is assumed that the DBMS can handle multiple simultaneous queries. If this is the case, then subqueries may be transformed into native format and queued for submission to the local DBMS by one task while output is formatted and queued for transmission to a D1* Node by another. Thus, processing of subqueries at the Data Node may proceed in "parallel" in the sense that each task simply responds to available input from its queue (like a "Data Flow Machine") rather than waiting for some prior task to signal it.

Subsystem 10: EXTRACT-AND-COMPUTE-DATA



Data Flow	Data Item	Source Function / Store	Destination Function / Store
D1	NDRL-subquery	Data-Node-Subquery-Queue	Receive Subquery
D2	NDRL-subquery	Receive Subquery	Translate-to-Native-Format
D3	native-subquery	Translate-to-Native-Format	Local-DBMS-Query-Facility
D4	raw-subresponse	Local-DBMS-Query-Facility	Receive Subresponse
D5	raw-subresponse	Receive Subresponse	Construct Virtual Key
D6	native-subresponse	Construct Virtual Key	Send Subresponse

Figure 3.10.3

Separate, parallel tasks are initiated for processing errors which may arise for each subquery which encounters problems. These tasks communicate, as appropriate, with the QUERY-MONITORING-AND-CONTROL Subsystem (QMC) at the D1* and D2Q/C and may either:

- Terminate the processing of the subquery.
- Return the subquery to a processing queue.

The action taken depends upon the directives received from QMC|D2Q/C.

3.10.3.4 Exception Conditions

The following errors may be encountered during the execution of this Subsystem:

- Subquery Translation Error.
- Subquery Submission Error (DBMS DML, DQL, or Report Writer error).
- Subresponse Absolute Quota Error.
- Subresponse User Quota Error.

The subject of handling quotas for subresponses is discussed in a paper on that topic which has been included in this IDN Subsystems Specification.

3.10.3.5 Rationale for Critical Decisions

The following critical decisions were made for this Subsystem:

1. Translation of the subquery to native format will be performed at the Data Node by special IDN software.

   - Although the database subschema resident at the D1* Node is sufficient for translation of the subquery from canonical form to NDRL, it is not adequate to provide all of the syntax peculiar to a particular DBMS.

   - Special software, resident at the Data Node, can be designed to supply this additional syntax and, essentially, emulate the role of a user submitting a subquery to the target database.

2. If possible, the actual execution of the subquery should be handled by the DQL or Report Writer of the local DBMS.

   - The necessity for developing software to directly access the database is eliminated.

   - Subqueries can be processed concurrently by the DBMS.

   - Error conditions can be detected and reported by the DBMS.

3. The subresponse will be generated in a "relational table format".

   - The subresponse can be conveniently formatted into messages.

   - Response composition can perform JOINs and SEMI-JOINs on the subresponses.

   - Subresponses can be JOINed with other databases (i.e., the subresponse from one subquery can be used as the input to another database subquery; this is a special case which may occasionally arise (see Note 3 in Section 3.10.6).

   - The final response can be conveniently handled by the user interface.

3.10.3.6 Open / Deferred Decisions

1. The decision was made to give the QMC the responsibility for the handling of those subresponses which must be used in a JOIN operation against either the source database or some other database (see Note 3, Section 3.10.6).

   This decision introduces some additional overhead into the IDN, but provides for a uniform handling of subqueries and subresponses by the Dl* and Data Nodes. This decision may need to be reviewed during design from an overall IDN performance and reliability perspective.

3.10.3.7 Critical Algorithms

The following algorithms are critical to the functioning of this Subsystem:

- Subquery Translation from NDRL to native format.
- Subresponse Formatting for transmission to Dl*.

The nature of both of these algorithms depends heavily upon the specific DBMS which manages the target database. A separate paper, of Subquery Processing Against the Basic Data Models, is included with the IDN Subsystems Specification and deal with this issue. The paper reviews the submission of subqueries and the receipt of responses from hypothetical DBMSs which use the following data models:

- Relational.
- Network.

- Hierarchical.
- Inverted File.

3.10.4 DATA REQUIREMENTS

3.10.4.1 Explanation of Major Data Flows

NDRL Subquery

This data flow is received by the current Subsystem in the form provided by the TRANSLATE-TO-NETWORK-DATA-REQUEST-LANGUAGE Subsystem.

Local-database-data

This data is held in the target database in a form consistent with the relational, network, hierarchical, or inverted file models.

Native-format-subresponse

This data is extracted from the local database in a format which is as consistent as possible with that of a relational table.

Abort/halt/continue-processing-message

The QMC may direct the Subsystem to either abort subquery execution, suspend execution until otherwise notified, or continue a suspended execution of a subquery.

Subquery-execution-error

The local QMC at the D1* Node is notified of any error which occurs during the translation or execution of the subquery.

Native-format-subquery

The subquery has been translated from NDRL to the native format of the local DBMS's DML, DQL, or Report Writer. All additional syntax required for query execution has been supplied.

Subresponse-size-message

QMC|D2Q/C is notified of the size of the subresponse.

Security-violation-message

The Security Administrators at the user and Data Nodes are notified of an attempt to submit an improperly authorized subquery against a sensitive database.

Subresponse-over-quota-message

If the size of the output exceeds the specified user quota or absolute quota then the appropriate action must be taken.

Native-format-subresponse

The response received from the database is transformed, as necessary, into relational table format and the virtual key is prefixed to each record.

3.10.4.2 Special Data Requirements

The EXTRACT-AND-COMPUTE-DATA Subsystem must have available to it all of the necessary syntactical and physical database information necessary for translation of subqueries from NDRL to native format.

It is assumed that the target databases are not the same as those used locally by the Data Node, they are, instead, copies or subsets of those databases which are maintained expressly for the purpose of network access.

Some databases may not be consistent with any of the four basic data models supported by the IDN. If this is the case, a version of the database which has been converted to one of the four standard data model will have to be maintained, or, the EXTRACT-AND-COMPUTE-DATA Subsystem will have to include special functions for handling the submission of subqueries to the special database and the acceptance of subresponses from it.

3.10.5 DATABASE REQUIREMENTS

3.10.5.1 Network Metadata

All requirements for network metadata are handled by the QMC.

3.10.5.2 Data Metadata

The Subquery control block provides all of the data metadata required by this Subsystem.

3.10.5.3 Other

None.

3.10.6 NOTES

1. The QMC must interact with this system at two levels:

   - Global: receiving Subresponse size information and sending abort / halt / continue messages.

   - Local: errors in Subquery processing requiring termination of SQCB at the D1*.

2. An audit trail of all Subquery execution errors should be maintained for the local Data Administrator to use in tracking the performance of the EXTRACT-AND-COMPUTE-DATA Subsystem and to provide indications of potential adjustments or improvements.

3. In the special case where two subqueries with the same parent query are directed against the same database, the subresponse from one query may be needed as input to the other. Such a case is recognized by QMC, and QMC transmits the subresponse needed for interrogation of the database to the D1*, which then passes it back to the Data Node for processing like any other subquery (or set of subqueries in the case of a JOIN or SEMI-JOIN). QMC also handles subresponses which must be used as input to subqueries against other databases.

3.11 SUBSYSTEM 11: PREPARE-OUTPUT-FOR-NETWORK

3.11.1 OVERVIEW

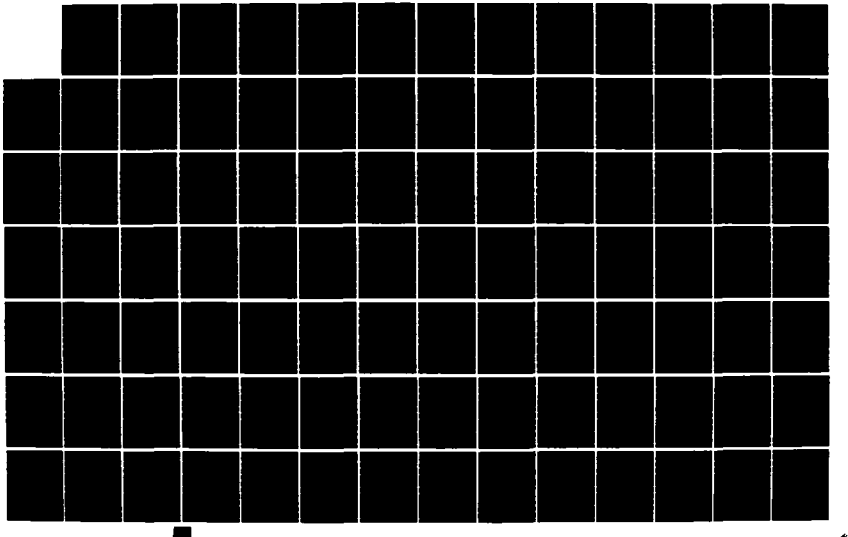
Subsystem 11, the PREPARE-OUTPUT-FOR-NETWORK Subsystem, resides at the D1* Nodes and must accept the subresponses passed to it by the previous Subsystem (at the Data Node), the EXTRACT-AND-COMPUTE-DATA Subsystem, and prepare them for the next Subsystem, the ASSEMBLE-COMPOSITE-RESPONSE Subsystem. The subresponses are already in relational table format, so this Subsystem may simply have to append a header block to the subresponse in order to process it.

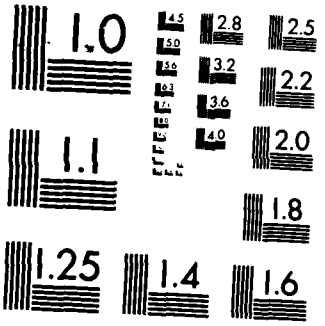
If any special data transformations are required in order to transform the native subresponse to the standard format, these are performed. Subsystem 11 also checks to be sure that a valid virtual key has been constructed and appended to each of the output records. This unique key is a code which indicates the source database for the subresponse and facilitates assembly of the composite response.

No errors should arise during the execution of this Subsystem. If a subresponse fails any of the simple checks which are made, then it is terminated and messages are sent to the QUERY-MONITORING-AND-CONTROL Subsystem (QMC) at the D1* Node.

AD-A173 344 DESIGN CONCEPTS FOR DATA BASE UTILITIES(U) AOG SYSTEMS 4/5
CORP HARVARD MA APR 86 RADC-TR-86-48 F30602-84-C-0011

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

3.11.2 INTERFACE SPECIFICATIONS

3.11.2.1 Intersubsystem Data Flows

Figure 3.11.1 shows the data flows between this and other Subsystems.

3.11.2.2 Subsystem Events

The event diagram for this Subsystem is shown in Figure 3.11.2.

3.11.3 TECHNICAL REQUIREMENTS

3.11.3.1 Actions

The PREPARE-FOR-NETWORK Subsystem resides at the D1* Node and receives subresponses in native format from the Data Node. These subresponses may already be consistent with the standard subresponse format or they may require some additional editing.

Additional editing may be required, depending upon the DBMS from which the subresponse was output, to make the subresponse in a form suitable for response composition. Examples of such editing are:

- Truncating trailing blanks.
- Separating data columns.
- Character set conversion (e.g. EBCDIC to ASCII).

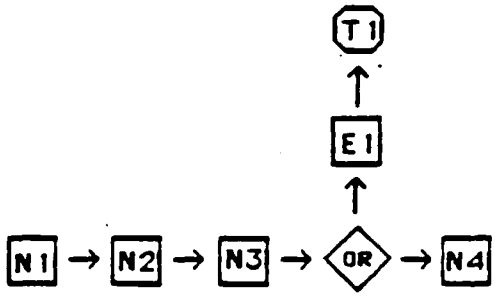
Subsystem 11: PREPARE-OUTPUT-FOR-NETWORK

In	Data Item	Source Subsystem
I1	Native-Format-Subresponse	EXTRACT-AND-COMPUTE-DATA

Out	Data Item	Destination Subsystem
O1	Standard-Format-Response	ASSEMBLE-COMPOSITE-RESPONSE

Figure 3.11.1

Subsystem 11: PREPARE-OUTPUT-FOR-NETWORK



Normal	Event Description
N1	Receive native format subresponse from Extract and compute data
N2	Perform any additional data transformations required
N3	Check subresponse format and virtual key
N4	Construct header block for subresponse and pass to next subsystem

Error	Event Description
E1	Error found in subresponse format or content

Terminal	Event Description
T1	Terminate subresponse processing due to detection of error

Figure 3.11.2

The subresponse is validated for correct format. If errors are encountered, they are reported to QMC at the D1* and the subresponse is terminated.

If the subresponse passes the validation step, it is given a header block containing information about the query and subquery which spawned the output and it is then queued for processing by the next Subsystem, the ASSEMBLE-COMPOSITE-RESPONSE Subsystem.

3.11.3.2 Intrasubsystem Data Flows

Figure 3.11.3 presents the intrasubsystem data flow for Subsystem 11.

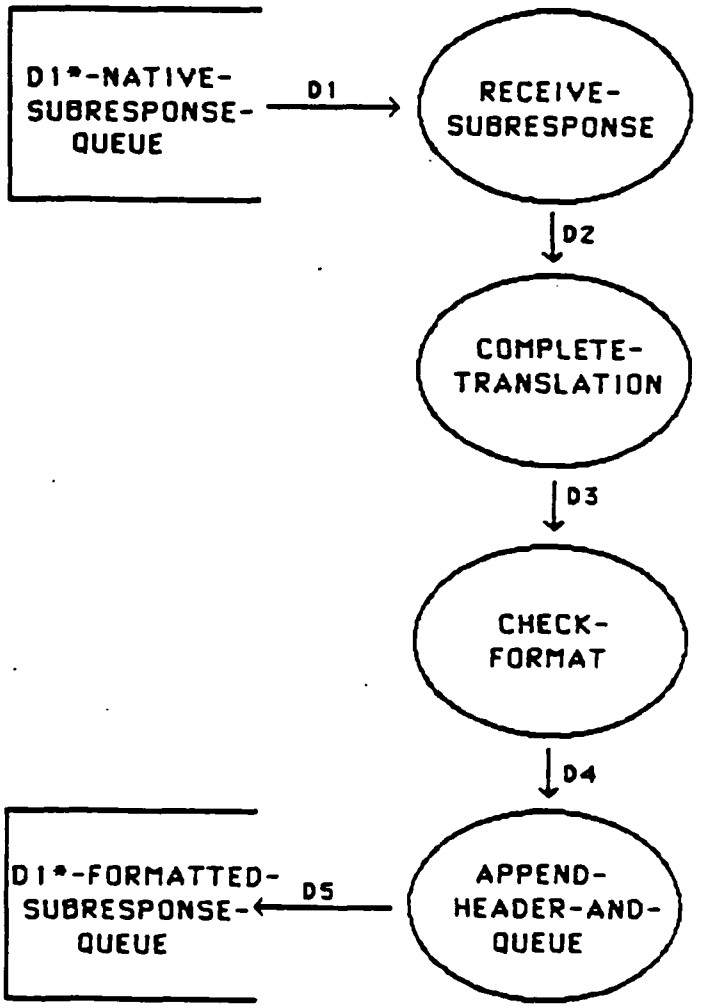
3.11.3.3 Tasking

The four major functions of this Subsystem operate serially for any given subresponse but are independent of one another. Therefore, separate queues and tasks may be maintained for each function in order to handle output preparation in a "parallel" manner. Additional tasks for any given function type may be initiated as required to handle varying volumes of subresponses.

3.11.3.4 Exception Conditions

Errors should be a rare occurrence in the preparation of output, but if inconsistencies are discovered in the formatted subresponse these will be reported to the local QMC (QMC|D1*), and the subresponse will be terminated. Since the IDN permits partial responses to queries, the termination of one subresponse does not effect the processing of other subresponses to the same query.

Subsystem 11: PREPARE-OUTPUT-FOR-NETWORK



Data Item	Data Item	Source Function / Store	Destination Function / Store
D1	native-subresponse	D1*-native-subresponse queue	receive-subresponse
D2	native-subresponse	receive-subresponse	complete translation
D3	standard-subresponse	complete translation	check format
D4	standard-subresponse	check format	append-header and queue
D5	standard-subresponse	append-header and queue	D1*-formatted sub-response queue

Figure 3.11.3

3.11.3.5 Rationale for Critical Decisions

1. The only critical decision made regarding the functioning of this Subsystem was the decision to have the Subsystem reside at the D1* Node rather than the Data Node.

   The rationale for this is as follows:

   - The overhead imposed on the Data Node by special IDN software should be minimized.

   - Any errors encountered can be more readily dealt with at the D1* Node.

3.11.3.6 Open / Deferred Decisions

1. An open decision at this point is whether or not it is possible to perform any summary calculations as part of output preparation. Considerations include:

   - For queries with multiple subqueries and subresponses, any required calculations will probably have to be deferred until a composite response is achieved. The earliest point at which the computations could be performed is at the conclusion of the ASSEMBLE-COMPOSITE-RESPONSE Subsystem processing.

   - As a matter of policy, it may be desirable to send detailed output to the DELIVER-COMPOSITE-RESPONSE Subsystem and make the D1 Node responsible for performing summary calculations. This would make both the detailed output and the desired computations available to the user.

   - For a few queries; e.g., those which require only simple computed values as the response, some communication overhead could be avoided by performing the computations within this

Subsystem or, in the ASSEMBLE-COMPOSITE-RESPONSE Subsystem, before shipping the output back to the D1 Node.

3.11.3.7 Critical Algorithms

The algorithm for checking the format of the subresponse, although straightforward, can be considered critical in that a variety of conditions might result in formatting errors in the subresponse. By detecting such errors in this subsystem, a cascade of error conditions in subsequent processing steps can be avoided.

3.11.4 DATA REQUIREMENTS

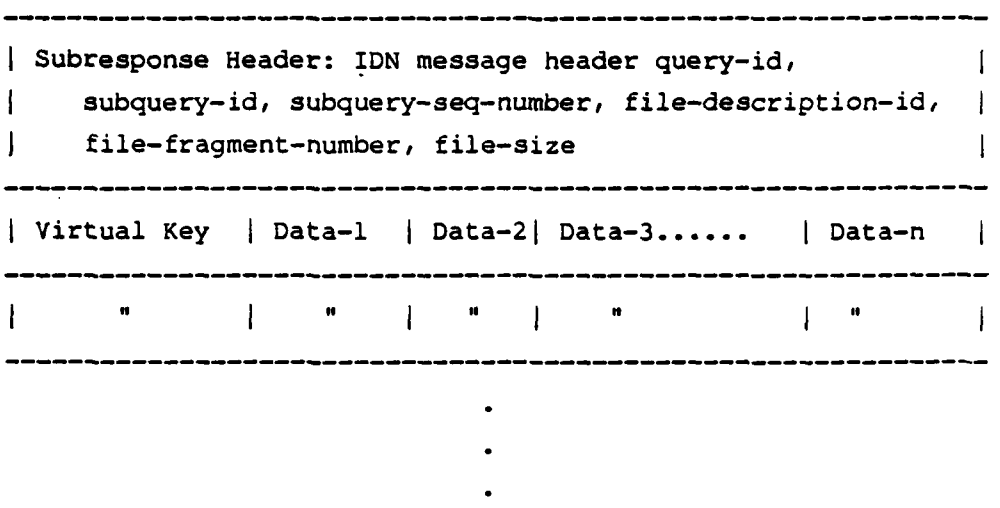
3.11.4.1 Explanation of Major Data Flows

Native-format-subresponse

The subresponse as returned from the EXTRACT-AND-COMPUTE Data Subsystem is in the format of the output received from the target database. This must be a relational table format. The major additions to the database output has been the prefixing of the virtual key for the target database to each output record, and a header which identifies the subquery which generated this response.

Standard-format-subresponse

The standard subresponse format is illustrated by the following:



The **subresponse header** contains essential control and tracking information necessary for verifying relationships among subresponses and subqueries, and for providing a "fail-safe" processing capability even if the QMC is disabled. This metadata includes:

- subquery-id - the unique identifier for the subquery.
- subquery-sequence - the processing sequence of the subquery relative to other subqueries for the same query (e.g., 1 of 3).
- File-description-id - the unique identifier for the subresponse.
- File-fragment-number - identifies the nth out of m file fragments in those cases where the subresponse must be divided into multiple messages.
- File-size - the size (in bytes and tuples) of the subresponse.

The virtual key is the unique code for the source database and is necessary for controlling response composition.

Data values are arranged in columns and have the following characteristics:

- All values are represented as ASCII codes (i.e., no binary formatted numeric fields).
- Each data value for a given data element is of a variable length. Data values are separated by a special field separator character. The correspondence between data values and elements is determined positionally.
- Text data values exceeding some predefined length limit (say, 72 characters) will be "folded" into multiple records and all associated data fields repeated in the new record so that a records will appear uniform without being exceedingly long. [Note: this characteristic depends upon the limitations which may exist within the IDN for the handling of long records. The system may be able to comfortably handle records of up to 32k bytes.]

3.11.4.2 Special Data Requirements

The special requirement for the handling of lengthy text data fields has been noted in the section above.

3.11.5 DATABASE REQUIREMENTS

3.11.5.1 Network Metadata

No network metadata is required for the preparation of database output.

3.11.5.2 Data Metadata

Some data metadata may be required if the output from a particular DBMS or database requires special reformatting in order to transform the subresponse into IDN standard form.

3.11.5.3 Other

None.

3.11.6 NOTES

1. It may be desirable to have the virtual key appended to each of the subresponse records as part of output preparation rather than as part of the EXTRACT-AND-COMPUTE-DATA Subsystem. Since the subresponse is likely to need some reformatting anyway, and since errors may be encountered in the format, it may be more efficient to defer the addition of the virtual key until this step.

(This page intentionally left blank)

3.12 SUBSYSTEM 12: IMPLEMENT-COMPOSITION-STRATEGY

3.12.1 OVERVIEW

Control of the standard format subresponses is passed from the PREPARE-OUTPUT-FOR-NETWORK Subsystem at a D1* Mode to the QUERY-MONITORING-AND-CONTROL Subsystem (QMC) at the Controlling D2Q Node (D2Q/C). QMC|D1* has sent the subresponse sizes for each subresponse to the IMPLEMENT-COMPOSITION-STRATEGY Subsystem at the D2Q/C Node. A given subresponse may, at this stage of its processing, remain at the current D1* Node for the particular subresponse, or be directed to an alternative D1* Node to be JOINed with another subresponse.

The subresponse sizes passed via the QMC provide Subsystem 12 with the means for determining the address for each subresponse and the order of response composition. Based upon the volume of the subresponses associated with a given query and available status information, the IMPLEMENT-COMPOSITION-STRATEGY Subsystem computes a response assembly strategy consistent with the processing sequence specified in the Query Execution Graph. The instructions for this strategy are in the Response Composition Program.

The QMC at the D2Q/C executes each step of this program by passing composition commands (SEMI-JOIN operations) to the copies of Subsystem 13, the ASSEMBLE-COMPOSITE-RESPONSE Subsystem, at the one or more D1* Nodes. In order to produce a composite response, subresponses are directed by QMC|D2Q/C to other D1* and D2Q Nodes as indicated by the Response Composition Program. The response composition process, therefore, may proceed in a parallel fashion depending upon the relationships between the subresponses.

This Subsystem also deals with null subresponses (i.e., subqueries which produce no subresponse due to their selection criteria or error occurrences). If the null subresponse was not critical to the formation of the response, then a response composition strategy is developed which provides a partial response to the original query.

If the destination node for a subresponse is determined to be unavailable for some reason, an alternate address is determined by Subsystem 12 and given, via QMC|D2Q/C and QMC|D1*, to Subsystem 13, the ASSEMBLE-COMPOSITE-RESPONSE Subsystem. If the primary and all alternate destinations are unavailable, then the subresponse may have to be terminated. Subresponses are queued for dispatch at the D1* Nodes after the assignment of their destination address. They remain in the queue until either receipt of the transmitted subresponse at the destination is acknowledged or a purge message is received from QMC at the D2Q/C.

3.12.2 INTERFACE SPECIFICATIONS

3.12.2.1 Intersubsystem Data Flows

Figure 3.12.1 shows the data flows between this and other Subsystems.

3.12.2.2 Subsystem Events

The event diagram for this Subsystem is shown in Figure 3.12.2.

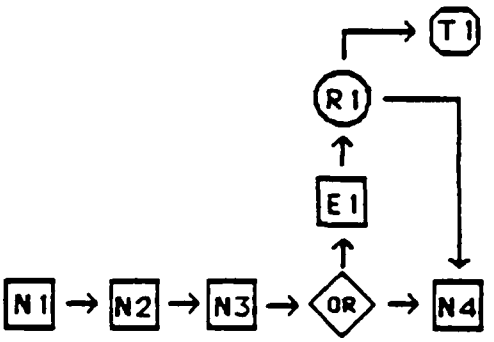
Subsystem 12: IMPLEMENT-COMPOSITION-STRATEGY

In #	Data Item	Source Subsystem
I1	Subresponse-size	QUERY-MONITORING-AND-CONTROL | D1*

Out #	Data Item	Destination Subsystem
O1	Response-composition-program	QUERY-MONITORING-AND-CONTROL | D2Q / C

Figure 3.12.1

Subsystem 12: IMPLEMENT-COMPOSITION-STRATEGY



Normal	Event Description
N1	Receive subresponse sizes from QMC | D1*
N2	Compute response composition strategy based on semijoins
N3	Generate response composition program and assembly node ids
N4	Pass response composition program to QUERY-MONITORING-AND-CONTROL at the D2Q / C

Error	Event Description
E1	Subresponse size not available for a subresponse

Response	Event Description
R1	QUERY-MONITORING-AND-CONTROL must purge the query if a key subresponse is unavailable

Terminal	Event Description
T1	Terminate due to unavailability of a key subresponse

Figure 3.12.2

3.12.3 TECHNICAL REQUIREMENTS

3.12.3.1 Actions

The formatted response is being held in a queue (the D1*-Formatted-Subresponse-Queue) by the previous Subsystem, the PREPARE-OUTPUT-FOR-NETWORK Subsystem (at a D1*), and the destination address for composition of the subresponse must be determined by Subsystem 12. Ultimately, the address is provided by the QMC but determination of the address is just one part of the overall Response Composition Program. This process is discussed in detail in a paper which accompanies the IDN Subsystems Specification: Using Joined-Subsets (SEMI-JOINs) in Subresponse Composition. The process can be summarized here by the following "case structure":

- CASE A: A given subresponse is needed for a JOIN operation with a database.

  The QMC at the D2Q/C has determined from the Query Execution Graph that the subresponse must be sent to a Data Node and provides the address of that node. When the new subresponse has been received from the Data Node, its size is passed by QMC|D1* and QMC|D2Q/C to the IMPLEMENT-COMPOSITION-STRATEGY Subsystem and a Response Composition Program is generated by Subsystem 12.

- CASE B: The subresponse is the only subresponse for the query.

  The IMPLEMENT-COMPOSITION-STRATEGY Subsystem directs that the subresponse remain at the D1* Node for conversion, by the ASSEMBLE-COMPOSITE-RESPONSE Subsystem, into "composite response format", and then be forwarded to the appropriate D1 Node for delivery to the user.

- CASE C: The subresponse is one of two or more subresponses which must be JOINed to produce a composite response.

  The IMPLEMENT-COMPOSITION-STRATEGY Subsystem must determine the communications cost (relative to operations on other subresponses) of either:

  - Performing SEMI-JOINs at the current D1* Node (and the JOIN at another D1* Node).
  - Performing SEMI-JOINs at another D1* Node (and the JOIN at the current D1* Node)
  - Performing JOINs at a designated D2Q Node (and, perhaps, SEMI-JOINs at the D1* Nodes).

Case "C" may be the most likely one for the IDN and, on average, a similar number of SEMI-JOINs are likely to be performed at all of the D1* Nodes. When two or more subresponses are generated for the same query, Case "C" may be viewed as executing "recursively" in that each SEMI-JOIN and JOIN operation at a given node produces a new subresponse which, in turn, must be JOINed with any remaining subresponses. This process continues until all subresponses have been JOINed into a single composite response.

The Response Composition Program which is generated by the IMPLEMENT-COMPOSITION-RESPONSE Subsystem consists of a series of simple relational operation and data transfer commands along with destination addresses for the subresponses, intermediate results, and final response. The operations and commands are:

- Perform SEMI-JOIN-select.
- Perform SEMI-JOIN-join.

- o Perform SEMI-JOIN-append.
- o Send output to designated node.
- o Transmit response to user.

In addition, QMC at the D2Q/C Node will send the following commands (via QMC at the D1* Nodes) in order to execute the Response Composition Program and deal with operational contingencies:

- o Cancel output.
- o Send output to (alternate) designated node.
- o Subresponse status request.

Once a destination address has been given to QMC|D2Q/C by Subsystem 12, QMC|D2Q/C must determine the availability of the destination node and queue the subresponse for transmission. If the destination is the current D1* Node, then no actual transmission will occur (a "virtual communications link" will transfer the subresponse from one queue to another within the D1* Node). If the destination is some other node, and the node is not available (e.g., a timeout has occurred), then QMC|D2Q/C must determine an alternative address and must requeue the subresponse.

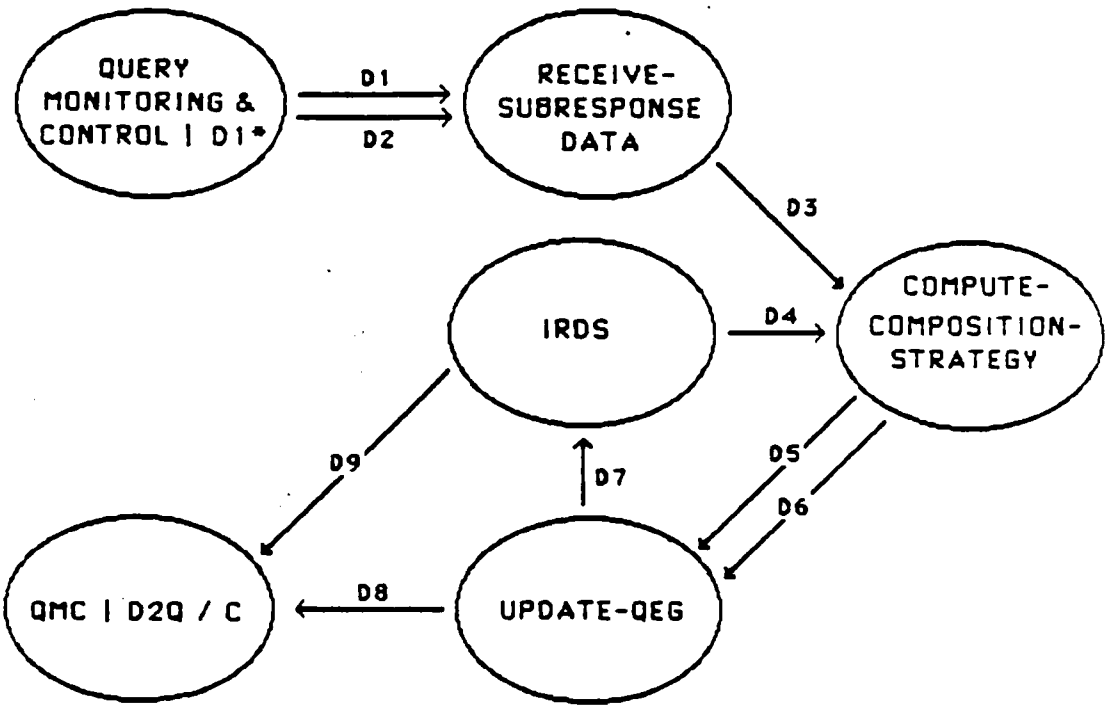
3.12.3.2 Intrasubsystem Data Flows

Figure 3.12.3 gives the intrasubsystem data flow diagram for Subsystem 12.

3.12.3.3 Tasking

Processing within this Subsystem can be viewed using the simple input, process, output model. Three independent tasks are initiated which

Subsystem 12: IMPLEMENT-COMPOSITION-STRATEGY



Data Flow	Data Item	Source Function / Store	Destination Function / Store
D1	subresponse-sizes	QUERY-MONITORING-AND CONTROL (D1*)	receive-subresponse-data
D2	D1*-node-ids	QUERY-MONITORING-AND CONTROL (D1*)	receive-subresponse-data
D3	subresponse-data	receive-subresponse-data	compute-composition-strategy
D4	QEG	IRDS	compute-composition-strategy
D5	updated-QEG	compute-composition-strategy	update-QEG
D6	composition-program	compute-composition-strategy	update-QEG
D7	updated-QEG	update QEG	IRDS
D8	composition-program	update QEG	QMC	D2Q / C
D9	updated-QEG	IRDS	QMC	D2Q / C

Figure 3.12.3

simply wait for appropriate input. The tasks may, therefore, run in parallel, driven by their internal queues.

3.12.3.4 Exception Conditions

1. Destination not available - request alternate destination.

2. Cancel output - if this is due to a low precedence level, the subquery will be restarted; if this is due to the unavailability of a subresponse composition node, then the subresponse will be terminated.

3.12.3.5 Rationale for Critical Decisions

1. The critical decision considered for this Subsystem is how to handle the dynamics of obtaining the destination addresses.

   As discussed in the presentation of cases, Section 3.12.3.1, this determination must be made through the use of cooperating Subsystems, i.e., Subsystem 12 and QMC at the D2Q/C and D1* Nodes, it cannot be accomplished by a single function operating independently.

3.12.3.6 Open / Deferred Decisions

None.

3.12.3.7 Critical Algorithms

The critical algorithm required for this process is the procedure for determining the response composition strategy. This is

discussed in a paper in Chapter 2: Using Joined-Subsets (SEMI-JOINs) in Subresponse Composition.

3.12.4 DATA REQUIREMENTS

3.12.4.1 Major Data Flows

Subresponse-sizes

The size of each subresponse in bytes. Included with the sizes must be the unique subquery, the query, the subresponse, and the node for the D1* Node where the subresponse is being held.

Response-Composition-Program

The sequence of relational operations and data transfer commands which must be executed by QMC for a response to be composed (included is an ordered list of the node identifiers at which the operations are to occur).

3.12.4.2 Special Data Requirements

Data must be available from the underlying protocol to indicate whether or not a subresponse has been transmitted to its destination successfully. The availability of a destination node must be readily determinable.

3.12.5 DATABASE REQUIREMENTS

3.12.5.1 Network Metadata

QMC at the D2Q/C must use network metadata to determine and assign destination addresses if any D2Q nodes used in the Response Composition Program.

3.12.5.2 Data Metadata

QMC at the D2Q/C must consult the Query Execution Graph to determine if a subresponse must be used in a subsequent Subquery.

3.12.5.3 Other

None.

3.12.6 NOTES

None.

(This page intentionally left blank)

3.13 SUBSYSTEM 13: ASSEMBLE-COMPOSITE-RESPONSE

3.13.1 OVERVIEW

Subsystem 13, accumulates all subresponses, completes the processing of the parent query, assembles the subresponses into a single composite response, and transmits the response to the companion D1 Interface Processor of the originating User Node. The ASSEMBLE-COMPOSITE-RESPONSE Subsystem resides at all D1* and D2Q nodes. It receives subresponses as determined by the Response Composition Program generated by Subsystem 12 and executed by the QUERY-MONITORING-AND-CONTROL Subsystem (QMC) at a D1* or a D2Q as required by circumstances.

This Subsystem must have some of the core functions of a Relational Database Management System (RDBMS) in order to properly complete query execution and to assemble the response. It may be necessary to compare key values in two or more subresponses, select qualifying records from a subresponse, and perform SEMI-JOINs and JOINs upon multiple subresponses. The result of all of these operations must be a meaningful composite response.

The ASSEMBLE-COMPOSITE-RESPONSE Subsystem may be directed to form subassemblies from subresponses at D1* Nodes before subresponses are sent to a final assembly point. This procedure is advantageous since "joined subsets" or("SEMI-JOINs") may be employed to reduce communications overhead due to response composition. The algorithm employed by Subsystems 12, 13, and QMC 'at the D2Q/C and D1* Nodes) to perform these functions is the Wong Algorithm. This diagnostic is described in a separate paper which is included with the IDN Subsystems Specification: Using Joined Subsets (SEMI-JOINs) in Subresponse Composition.

This Subsystem receives messages from the QMC at the D2Q/C. These messages direct the SEMI-JOINs across the D1* Nodes which have subresponses for a given query. The usual case is that virtually all of the response composition process, including the formatting of the final result before sending it on the the D1 Node and the user , takes place at D1*. Nodes If one or more of the D1* Nodes cannot handle the processing load or the volume of the resultant response, then a D2Q Node will be used to complete the assembly.

Response composition is facilitated by the virtual key which is carried by each subresponse. This key makes it possible to perform relational operations upon tables (subresponses) from different source databases as if they were all from the same homogeneous database. Subsystem 13 automatically executes generated command sequences against the "database" of subresponses using the source information in the virtual key.

A variety of error conditions may arise during the processing and assembly of the subresponses for a particular query. A common condition is the production of more output records than a user's quota (or the system quota) allows. If this or any other error arises, the Subsystem will attempt to salvage at least a partial response for the user.

3.13.2 INTERFACE SPECIFICATIONS

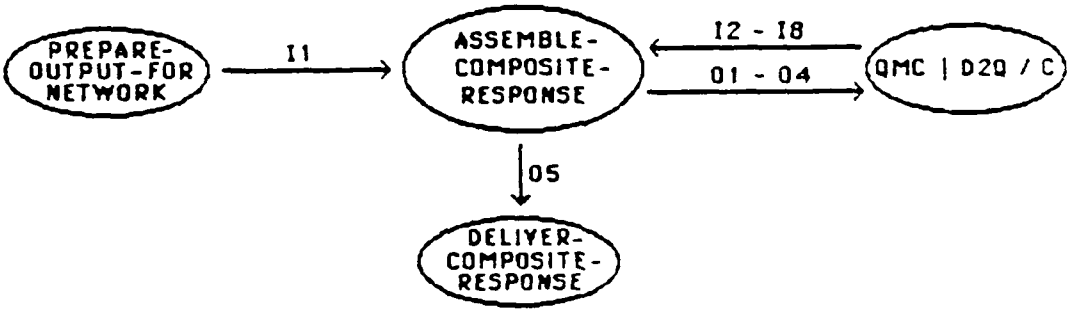
3.13.2.1 Intersubsystem Data Flow Diagram

Figure 3.13.1 shows the data flows between this and other Subsystems.

3.13.2.2 Subsystem Events

The event diagram for this Subsystem is shown in Figure 3.13.2.

Subsystem 13: ASSEMBLE-COMPOSITE-RESPONSE

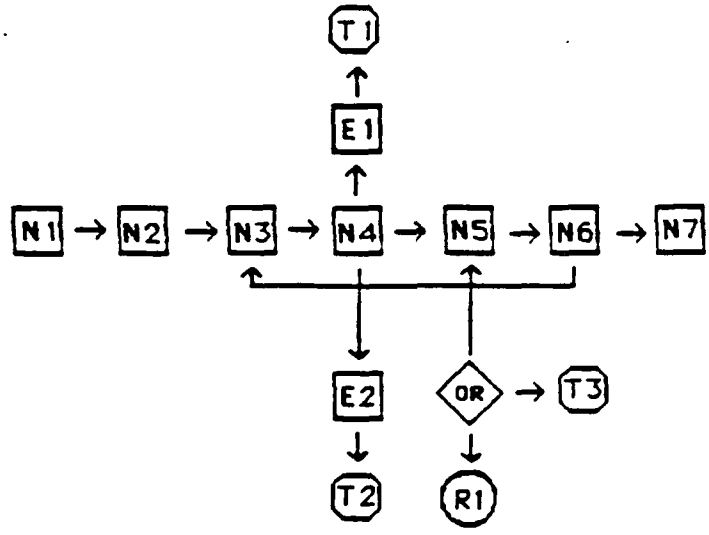


In	Data Item	Source Subsystem
I1	formatted-subresponse	Prepare Output for Network
I2	perform-semijoin-select	Query Monitoring & Control|D2Q/C
I3	perform-semijoin-join	Query Monitoring & Control|D2Q/C
I4	perform-semijoin-append	Query Monitoring & Control|D2Q/C
I5	send-output-directive	Query Monitoring & Control|D2Q/C
I6	cancel-output-directive	Query Monitoring & Control|D2Q/C
I7	subresponse-status-request	Query Monitoring & Control|D2Q/C
I8	transmit-response-to-user	Query Monitoring & Control|D2Q/C

Out	Data Item	Destination Subsystem
O1	response-composition-error	Query Monitoring & Control|D2Q/C
O2	subresponse-status-data	Query Monitoring & Control|D2Q/C
O3	output-transmission-error	Query Monitoring & Control|D2Q/C
O4	response-composition-complete	Query Monitoring & Control|D2Q/C
O5	composite-response	Deliver Composite Response

Figure 3.13.1

Subsystem 13: ASSEMBLE-COMPOSITE-RESPONSE



Normal	Event Description
N1	Receive response assembly instructions from controlling D20/C
N2	Receive addressed subresponse from D1*
N3	Receive second (if any) subresponse for composition
N4	Perform composition operation on two subresponses
N5	Format resultant subresponse
N6	Determine if all subresponses processed for current query
N7	Format composite response for transmission to D1

Error	Event Description
E1	User output quota exceeded
E2	Error in performing composition operation

Response	Event Description
R1	QUERY-MONITORING-AND-CONTROL requests subresponse status

Terminal	Event Description
T1	Terminate due to excessive output
T2	Terminate due to composition error
T3	Terminate due to unavailability of required subresponse

Figure 3.13.2

3.13.3 TECHNICAL REQUIREMENTS

3.13.3.1 Actions

The actions performed by this Subsystem are simple when taken individually, but become complex when considered in the aggregate. All actions performed by the ASSEMBLE-COMPOSITE-RESPONSE Subsystem are directed by QMC at the D2Q/C Node via QMC at the D1* Node.

QMC determines which operations to perform and their sequence from the Response Composition Program which was generated for the query by the IMPLEMENT-COMPOSITION-STRATEGY Subsystem (Subsystem 12).

The basic sequence of events for Subsystem 13 is as follows:

- o A message is received from QMC to perform an operation on a subresponse or subresponses. The message will include one or more of the following commands:
  - Perform-SEMI-JOIN-select.
  - Perform-SEMI-JOIN-join.
  - Perform-SEMI-JOIN-append.
  - Send-output (to designated node).
  - Cancel-output.
  - Send-subresponse-status.
  - Transmit-response-to-user.

- The Assemble-Composite-Response Subsystem checks the D1*-Formatted-Subresponse-Queue for the subresponse or subresponses that are needed to perform the required operation.

- When the required subresponses are available, the operation prescribed by QMC is carried out.

- The resultant subresponse is formatted and returned to the D1*-Formatted-Subresponse-Queue.

- The ASSEMBLE-COMPOSITE-RESPONSE Subsystem repeats this cycle until all subresponses for the query have been merged into a single composite response.

- Once the composite response has been achieved, a message is sent to QMC|D2Q/C (via QMC|D1*) which, in turn, directs that the response be sent to the D1 Node for delivery to the user.

It is important to remember that the ASSEMBLE-COMPOSITE-RESPONSE Subsystem resides at all D2Q and D1* Nodes. Depending upon the relationships among the subresponses for a given query, the above actions may occur in parallel at several D1* Nodes. Each resultant subresponse is JOINed with another resultant subresponse for the same query. This "recursive" composition process finally produces a composite response when there are no more subresponses to be JOINed. Normally, all composition is performed at D1* nodes. However, if a participating D1* Node cannot handle the processing load required by a phase of the assembly process or if the D1* Node cannot handle the volume of the resultant subresponse, then a D2Q Node may be used to complete the response assembly.

3.13.3.2 Intrasubsystem Data Flows

Figure 3.13.3 gives the intrasubsystem data flow diagram for Subsystem 13.

3.13.3.3 Tasking

As indicated in Section 3.13.3.1, the ASSEMBLE-COMPOSITE-RESPONSE Subsystem executes as separate tasks or processes on each D2Q and D1* Processor. QMC coordinates their execution allowing them to process a given set of subresponses with as much parallelism as possible. This Subsystem effectively turns the IDN into a large distributed processing system capable of executing either serially or in parallel as required.

3.13.3.4 Exception Conditions

The following exception conditions may arise:

1. Unable-to-transmit error - an output cannot be transmitted to the designated destination node.

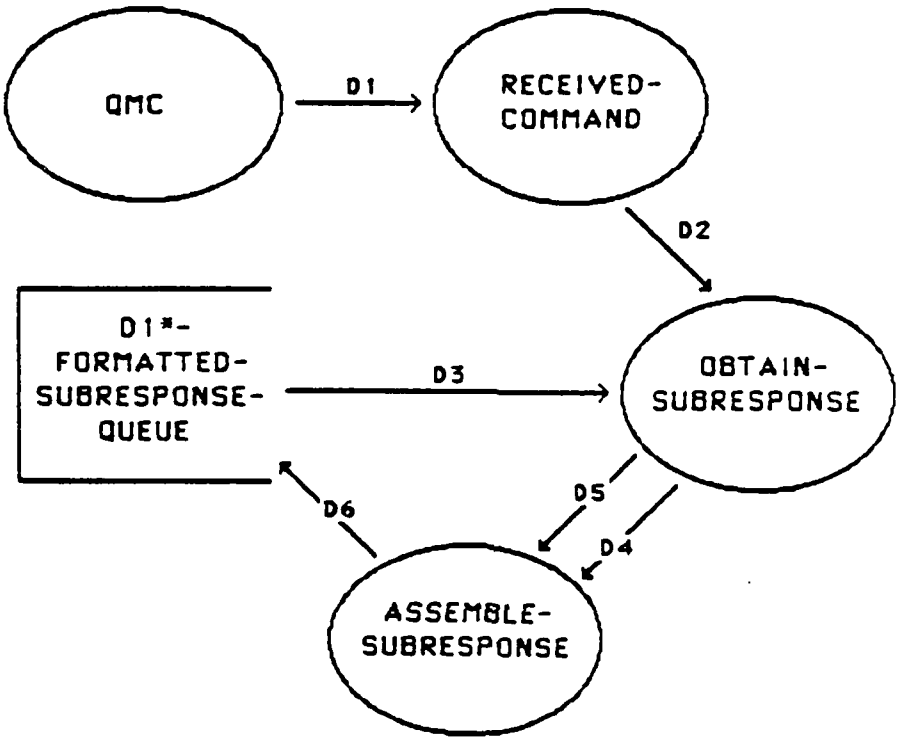
2. Response composition error - an error, one of several possible types, has occurred during the response composition process. Composition of the input subresponses cannot be completed.

3. Subresponse over quota error - the threshold set for any single subresponse for the given user and precedence level has been exceeded. Only a partial response can be returned to the user.

3.13.3.5 Rationale for Critical Decisions

1. The critical decision for this Subsystem was to decision to have QMC drive its execution using the Response Composition Program

Subsystem 13: ASSEMBLE-COMPOSITE-RESPONSE



Data Item	Data Flow	Source Function / Store	Destination Function / Store
D1	composition-command	QUERY-MONITORING-AND-CONTROL	receive command
D2	assembly-order	receive command	obtain subresponse
D3	formatted-subresponse	D1*-formatted subresponse queue	D1*-formatted sub-response queue
D4	assembly-order	obtain subresponse	assemble sub-response
D5	formatted-subresponse	obtain subresponse	assemble sub-response
D6	assembled-subresponse	assemble subresponse	D1*-formatted sub-response queue

Figure 3.13.3

generated by the IMPLEMENT-COMPOSITE-STRATEGY Subsystem rather than follow some preset (invariant) composition algorithm.

By using this approach, the dynamic nature of the subresponses (e.g., relative time of generation, relative size, and null subresponses) can be effectively dealt with. An added benefit of this decision is that subresponse composition can proceed in parallel using all of the participating D1* Nodes along with the D2Q Nodes as coordinated processors.

3.13.3.6 Open / Deferred Decisions

None.

3.13.3.7 Critical Algorithms

The Wong SEMI-JOIN algorithm is key to the operation of this Subsystem. It is presented in detail in a separate paper which accompanies the IDN Subsystems Specification: Using Joined-Subsets (SEMI-JOINs) in Subresponse Composition.

3.13.4 DATA REQUIREMENTS

3.13.4.1 Major Data Flows

Composition-message

A directive from QMC to perform a composition operation. The message may include an assembly-node-id. This id identifies the node to which output must be sent for the next step in response composition.

3.13.4.2 Special Data Requirements

The underlying protocol must be capable of informing the Subsystem when a transmission cannot be completed due to unreliable communications.

3.13.5 DATABASE REQUIREMENTS

3.13.5.1 Network Metadata

The QMC Subsystem must use both static and dynamic network metadata to determine appropriate nodes and to assign destination addresses.

3.13.5.2 Data Metadata

The QMC Subsystem must consult the Query Execution Graph to determine if a subresponse must be used in a subsequent subquery.

3.13.5.3 Other

None.

3.13.6 NOTES

None.

3.14 SUBSYSTEM 14: DELIVER-COMPOSITE-RESPONSE

3.14.1 OVERVIEW

Subsystem 14 completes the processing of the query and its response (as far as the IDN processors are concerned) by accepting the composite response from the D1* or D2Q node at which it was assembled, formatting it for presentation, and transmitting it to the User Interface at the User Node.

The composite response is received by the Subsystem in a form which appropriate for internal IDN transmission, but which may not be appropriate for the User Interface at the User Node.

- All NANS for data elements must be translated back to the original LUNS names used when the query was submitted.

- The screen mapping information which was passed along with the query to the VALIDATE-TOKENIZED-QUERY Subsystem and saved by that Subsystem in the QCB must be properly associated with the response output stream.

- It may also be necessary to perform summary calculations against the response data. Such calculations would be performed as part of response composition if the calculated results alone were required by the user. If both the computations and the detail records are required, however, then the calculations must be performed at the D1 Node prior to transmission of the total response to the user.

Once the response has been properly formatted and transmitted to the User Node, the DELIVER-COMPOSITE-RESPONSE Subsystem issues a "response-

delivered-to-user" message to the QUERY-MONITORING-AND-CONTROL Subsystem (QMC) at the D1 Node. This QMC process, in turn, communicates with the QMC processes at all other nodes participating in the processing of the query in order to purge all QCBs, SQCBs, the QEG, and all subresponses from the IDN.

If the user is unavailable to receive the response and the User-node cannot hold the response, the response is saved at the D1 Node (temporarily) in a queue. The response will be held on the queue until the user requests transmission, or until the queue is filled. In event of a full queue, responses will be deleted based on age. The D1 node administrator will be warned when this queue reaches a critical percentage of being full, and when responses are purged.

3.14.2 INTERFACE SPECIFICATIONS

3.14.2.1 Intersubsystem Data Flows

Figure 3.14.1 shows the data flows between this and other Subsystems.

3.14.2.2 Subsystem Events

The event diagram for this Subsystem is shown in Figure 3.14.2.

3.14.3 TECHNICAL REQUIREMENTS

3.14.3.1 Actions

The actions taken by this Subsystem may be summarized as:

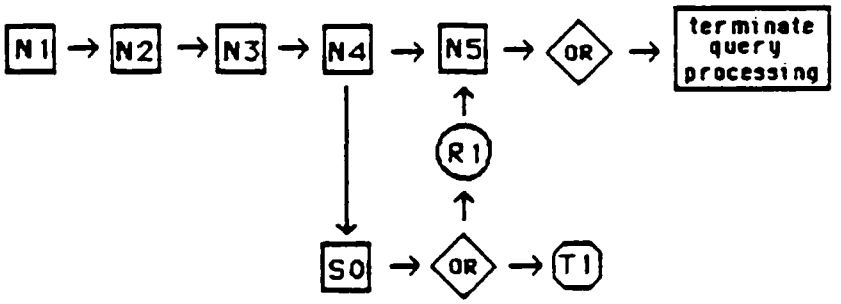
Subsystem 14: DELIVER-COMPOSITE-RESPONSE

In	Data Item	Source Subsystem
I1	Composite-Response	Assemble-Composite Response

Out	Data Item	Destination Subsystem
O1	Formatted-Response	User-Interface

Figure 3.14.1

Subsystem 14: DELIVER-COMPOSITE-RESPONSE



Normal	Event Description
N1	Receive composite response from D2Q
N2	Perform final response processing operations
N3	Determine if User is available to receive response
N4	Queue response for transmission to D1
N5	Send response-delivered message to QMC | D1

Special	Event Description
S0	User is not available. Save response (temporary)

Response	Event Description
R1	User Node Administrator requests transmission of saved response

Terminal	Event Description
T1	Saved response expires

Figure 3.14.2

- Receive composite response.
- Perform formatting operations.
- Transmit response to User Node.

If the composite response is not received within a "reasonable" period of time, either the User or the User Node Administrator may issue a request for the processing status of the query. QMC at the D1 Node must consult the QCB for the query to determine the D2Q Controlling Node (D2Q/C). QMC at the D2Q/C must consult the QEG and Response Composition Program for the query as well as the QCB and SQCBs in order to determine processing status.

Once status is determined, the user may have the authority to increase the precedence level of the query, or may decide to terminate the query. In either case, this information is communicated to QMC|D2Q via QMC|D1 and the query processed accordingly.

Once a composite response has been received by the DELIVER-COMPOSITE-RESPONSE Subsystem, at the D1 Node, one or more of the following formatting operations must be performed:

- The response must be blocked and formatted into "IDN standard output format" for transmission to the User Interface at the User Node.
- If the response is incomplete (i.e., a partial response), appropriate null values are inserted into the response and a header message is inserted indicating which data element values were not available. For security reasons, the exact source(s) of the missing values will not be identified.
- If screen mapping information is available (saved in the QCB by the RECEIVE-TOKENIZED-QUERY Subsystem), this information will be inserted at the appropriate points in the response

stream in order to facilitate output display by the User Interface at the User Node.

In addition, calculations may have to be performed on the detail records of the output and the results included in the response stream before it is transmitted to the user. Such calculations (from the perspective of a two dimensional report) could include:

- Row and Column totals.
- Row and Column averages (means).
- Percentages of one row or column with respect to another.
- Arithmetic operation across two rows or two columns.
- Summary computations at control breaks.

The final response may also have to be sorted before it is transmitted to the user. This would be required if control breaks of any kind are included in the report.

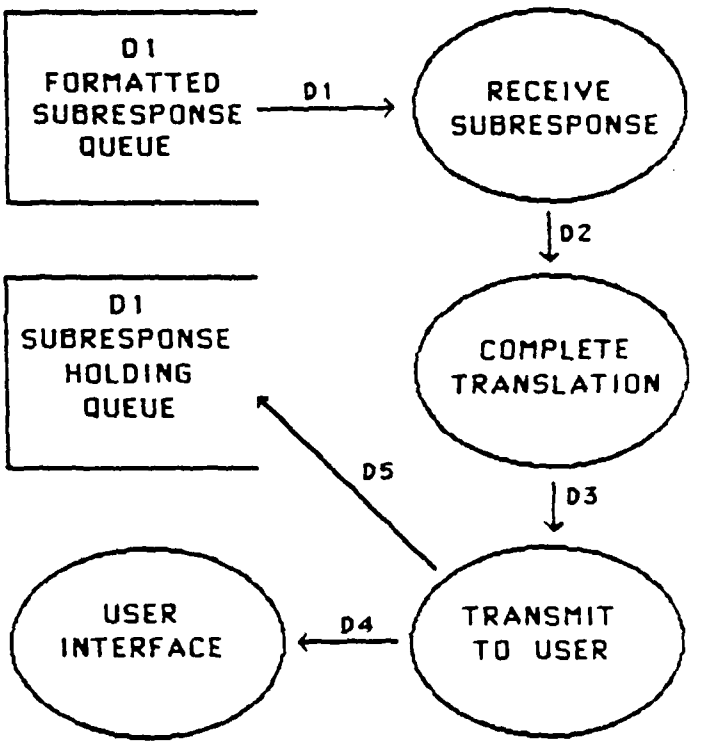
3.14.3.2 Intrasubsystem Data Flows

Figure 3.14.3 gives the intrasubsystem data flow diagram for Subsystem 14.

3.14.3.3 Tasking

As with most of the IDN Subsystems, the DELIVER-COMPOSITE-RESPONSE Subsystem is entirely data flow driven. One task constantly checks the input queue for a composite response while another task waits for a

Subsystem 14: DELIVER-COMPOSITE-RESPONSE



Data Item	Data Flow	Source Function / Store	Destination Function / Store
D1	formatted subresponse	D1-formatted sub-response queue	receive subresponse
D2	formatted subresponse	receive subresponse	perform computations
D3	computed subresponse	perform computations	transmit to user
D4	complete subresponse	transmit to user	user-interface
D5	complete subresponse	transmit to user	D1-formatted sub-response queue

Figure 3.14.3

signal to process (format and compute) the composite response. Finally, a third task checks the output queue to determine if there are any completed responses to be sent to the user.

3.13.3.4 Exception Conditions

The major exception conditions which may arise are:

1. No composite response has been received.

2. A partial response has been received.

3. Response cannot be properly formatted.

4. Computations cannot be performed.

5. User Node (user) is unavailable to receive response.

3.14.3.5 Rationale for Critical Decisions

The decision to perform summary calculations at the D1 just prior to transmission of the response to the user must be considered with this Subsystem.

This decision is justified if and only if the user wishes to receive both the computed results and the detail records. If only the computed results are requested, then the calculations will be performed by the ASSEMBLE-COMPOSITE-RESPONSE Subsystem as early in the response composition process as possible in order to reduce communications overhead.

3.14.3.6 Open / Deferred Decisions

None.

3.14.3.7 Critical Algorithms

Only simple formatting and computational algorithms are required.

3.14.4 DATA REQUIREMENTS

3.14.4.1 Major Data Flows

Composite-response

The IDN internal form of the query response.

Complete-response

The IDN standard format response as it is transmitted to the User Node.

3.14.4.2 Special Data Requirements

None.

3.14.5 DATABASE REQUIREMENTS

3.14.5.1 Network Metadata

QMC at the D1 and D2Q/C may have to use network metadata in order to determine the status of processing for a particular query.

3.14.5.2 Data Metadata

QMC at the D2Q/C must consult the QCBs, QEG, SQCBs, and Response Composition Program in order to determine the status of a particular query. This information is also used in order to purge the completed query from the IDN.

3.14.5.3 Other

None.

3.14.6 NOTES

None.

CHAPTER 4

IDN MANAGEMENT UTILITIES

4.1 INTRODUCTION

As specified in Chapter 3, Subsystems Specification, the IDN is a large and sophisticated system. This chapter discusses the requirements for utilities to create, operate and maintain the IDN.

4.1.1 Approach

Two basic environments are supported by the IDN Management Utilities, which will be described:

- The Support Environment.
- The Operational Environment.

Special consideration is given to requirements which are common to both environments, and those which require an interface between the environments.

Outstanding issues for each environment are discussed.

4.1.2 Definitions

As stated in earlier chapters, the IDN consists of existing Data Nodes and User Nodes, and specialized dedicated to support query processing. The specialized nodes are of three types: D1; D1*, and D2Q. All nodes of a given type share a common hardware architecture.

The term "IDN Software" is used to refer to the implementation of the IRDS functionality referenced, the query processing functions defined in Chapter 3, and those utilities identified in this chapter as residing at these specialized nodes. This definition excludes the actual queries which are run, which may be regarded as "applications" by the IDN Software. This definition does not exclude the possibility that part or all of the IDN system software may be implemented in firmware or in programmable read-only memories (PROMs). It is also assumed that, from a security point of view, the IDN Software is of a high degree of trustedness.

In addition to the IDN Software, operation of the IDN also requires the availability and control of metadata. IDN metadata is defined to be the contents of the dictionaries at the various IDN nodes. This metadata includes the static network view, LUNSs, CDRs, and IDN-Database-Views. An IDN-Database-View is the representation in a DI* dictionary of the Subschema of a database at a Data Node which defines which data is accessible by the IDN.

4.1.3 User Communities

For purposes of this chapter, the following groups of people are regarded as distinct communities:

- The users of the IDN.

  The users of the IDN are the personnel who submit queries and use the information produced by them.

- The IDN Support Group.

  This group is charged with the following responsibilities:

  - Development and maintenance of the IDN Software.

- Definition and modification of all CDRs and their mappings to IDN-Database-Views.

- Distribution of IDN Software.

- Distribution of CDRs to D2Q and D1* Nodes.

- Distribution of subsets of the NANS which are required at the D1 Nodes.

- Control of the hardware configuration of individual nodes, which may vary according to the discretion of the responsible administrators.

- Development and maintenance of the facilities to meet the required support responsibilities.

The facility at which the IDN Support Group operates will be referred to as the IDN Support Facility (IDNSF).

o IDN Operations Management.

This group consists of the administrators who oversee the day-to-day operation of the IDN.

The administrators include:

- User Node Administrators.

- D1 Node Administrators.

- D2Q Node Administrators.

- D1* Node Administrators.

- Data Node Administrators.

- IDN Administrator.

Chapter 3, The Subsystems Specification, has addressed the requirements of the first group. This chapter addresses the requirements of the last two groups.

4.2 UTILITY REQUIREMENTS

In this section the requirements of both the IDN Support Group and IDN Operations Management are identified.

4.2.1 IDN Support Group Requirements

The IDN Support Group provides support to both IDN users in their use of the facilities of the IDN and to IDN Operations Management in running and managing the IDN. For the purpose of analyzing utility requirements, the IDN support group may be considered as providing two basic services to these communities:

- o Development and Maintenance Support of the IDN Software.

- o Global Metadata Management and Distribution Control.

Both of these types of services will have to be provided in an environment where strict configuration management and control for both IDN Software and metadata exists.

These services require distinct technical capabilities. The first category is related to software engineering capabilities; the second one is analogous to data analysis and data administration services. Configura-

tion management and control is required to assure the integrity of the IDN. These services will now be discussed in more detail.

4.2.1.1 Software Development and Maintenance Requirements

It may be assumed that an initial version of the IDN Software will be made operational at the beginning, and which will be enhanced at a later time. Since initial development and maintenance and ongoing development involve significant differences in problems to be addressed, they are addressed separately.

4.2.1.1.1 Initial Development

Assurance must exist that the version of the IDN Software produced initially will consist of a viable system, and that a clear progression path to later versions exists.

The security requirements of the IDN hardware/software architecture imply stringent controls on the development environment a the IDNSF. Those controls include:

- Conformance with the "Orange Book" requirements for the required level of trustedness.
- Extensive monitoring of the effort.
- Strict configuration controls.

Upon completion of the initial development, the IDN must be "created". From the IDN Support Group's perspective, this involves installing the systems at various nodes. The installation process must ensure that no customization of the system is allowed which would violate the security and integrity of the IDN.

4.2.1.1.2 Ongoing Development and Maintenance Requirements

There are two types of ongoing development:

- The development to incorporate those Data Nodes and User Nodes with hardware/software configurations which could not be addressed in the initial development.
- The development to incorporate deferred features or enhancements identified by IDN users and operations management.

Ongoing development has all the requirements of initial development. In addition, ongoing development and maintenance require the development of plans of action for phasing of modifications.

Maintenance consists of correcting problems encountered in the day-to-day operation of the IDN. The problems which an individual IDN user or administrator encounters may not be limited to the IDN Software, but may result from insufficient documentation or training.

By nature, user liaison and maintenance activities are event-driven. This implies a different form of activity tracking than for a development project. In a development project, activity is tracked against a plan. User liaison and maintenance require an allocation of resources for to handle problems as they occur. Time and resources required to correct problems or provide other forms of technical assistance must be associated either by the user needing support or by problem, or both. The information on problems and other requests for technical assistance must be analyzed to determine if there is any pattern within the requests for service. If so, this is used in planning new subsequent development, documentation, or training projects.

Maintenance implies facilities for problem analysis and debugging. Security restrictions can present special problems to those responsible for error detection and correction. Security attributes associated with

data in the operational system will generally prohibit maintenance personal from dealing directly "in the field" with certain problems. File or memory dumps which are normally used for debugging could cause a breach of security.

After completion of ongoing development projects and corrections to software, documentation, and training materials which were performed in maintenance, there is a need to distribute the the changes to the field. This implies a system to maintain an inventory of "who has what" and control distribution of software and other deliverables. This system must keep track of not only what was shipped, but it must assure receipt of the deliverables by the appropriate organization, and in the case of the software identify that the software has been installed and is in use.

It must be recognized that security requirements on the IDN Software, and the resultant need for verification of software corrections, may conflict with the IDN Support Group's ability to provide timely services in the area of software maintenance.

4.2.1.2 Global Metadata Management and Distribution Control

The IDN Support Group is responsible for:

- Collection of the proposed IDN-Database-Views for Data Node Databases.
- Consultation with Dl* and Data Node Administrators, as required, to provide IDN-Database-View which can support queries.
- Definition of CDRs, mappings, and DPP relationships.
- Coordination in activating new IDN-Database-Views.

- Distribution of metadata:
  - CDRs to D2Q nodes
  - CDRs and IDN Database Views to D1* Nodes
  - CDR-ELEMENTS to D1 Nodes
- Maintenance of the correspondence between security classifications and CDR-ELEMENTS.
- Maintenance of the distribution inventory of CDRs, IDN-Database-Views, and CDR-ELEMENTS and security classifications for users supported by D1 Nodes.

4.2.2 IDN Operations Management Requirements

The architecture of the IDN is based on three different types of processors, each of which performs different functions. The three processor types have been designated D1, D2Q, and D1* respectively. Given the differentiation of processor roles, the individuals responsible for each of the processors will also have differing roles. The following section considers the following:

- Administration of D1 Nodes
- Administration of D2Q Nodes
- Administration of D1* Nodes
- Administration of the IDN
- Implementing Change Within the IDN

4.2.2.1 The D1 Processor

The primary function of the D1 processor within the IDN is twofold:

- It is the entry point by which a user node is connected to the IDN. As such, all user queries go to a D1 node where they receive first order validation.

- It supports the development and maintenance of user queries. The D1 processor fulfils this by containing the dictionary which holds both user queries and the various Local User Name Spaces (LUNS) for the different user communities at the User Node.

The role of an administrator of a D1 Node is primarily in support of the user communities at all the User Nodes which the given D1 Node services. As such, a D1 administrator must perform the following functions:

- Define and maintain all user profiles in the D1 dictionary.

- Define and maintain the LUNS associated with each user.

The D1 Administrator responsibilities, identified above, require the following IDNSF support utilities:

- The maintenance of user profiles does not require any specific IDNSF functionality, and can be handled entirely through the normal maintenance facilities.

- The IDNSF must provide special support for the D1 Administrator at the time a LUNS is to be defined or modified.
  A D1 Administrator maintains a LUNS by identifying the need for

  - creating or deleting entities of type LOCAL-ELEMENT-NAME in the D1 dictionary, and

- creating or deleting relationships from LOCAL-ELEMENT-NAMEs to CDR-ELEMENTs.

Each user's security classification is defined in the D1 dictionary, but the security classification of each CDR-ELEMENT is not available in that dictionary. Without special support from the IDNSF, it would be possible for the D1 administrator to assign a given user a LOCAL-ELEMENT-NAME for a CDR-ELEMENT which the user is not authorized to see. Since this is not acceptable, the IDNSF must:

- Prevent the D1 Administrator from defining a LUNS which would violate security; in fact, the IDNSF must coordinate all changes to security information.

- Notify the D1 administrator of the need to revalidate those effected LUNSs when the security classification of a CDR-ELEMENT is changed.

- Provide new CDR-ELEMENTs which are used to update the D1 dictionary.

- Identify obsolete CDR-ELEMENTs for deletion from the D1 dictionary.

- Notify the D1 Administrator as to when queued changes are to take effect.

o Software and/or hardware upgrades require close cooperation between a D1 Administrator and the IDNSF.

4.2.2.2 The D2Q Processor

The D2Q processor has two primary responsibilities in the processing of user queries:

- Query validation is completed at a D2Q processor. In validating the query, it decomposes the query into canonical subqueries, and chains the Subquery Control Blocks (SQCBs) together to form the Query Execution Graph (QEG).

- A D2Q processor controls the execution of the query. After the QEG is generated, the D2Q assigns nodes to process subqueries and dispatches those subqueries to D1* nodes for processing. Based upon subquery response sizes, a D2Q processor determines where subsequent subqueries must be executed and determines the destination nodes for subresponses.

The D2Q nodes also play an import role in query recovery. Each D2Q node may have several shadow D2Q nodes. Likewise, each D2Q node may act as a shadow for several other D2Q nodes. In the shadow role, a D2Q node retains a copy of the Query Control Block generated at another node. The shadow node maintains the QEG in parallel with the primary D2Q node. Thus, if the primary node should fail, the shadow node is able to take over control of the query quickly.

Since the D2Q node operates without interface to any user, the D2Q Administrator's role can be viewed as primarily technical. In particular, the D2Q Administrator is responsible for applying changes to the D2Q dictionary. This dictionary contains the Coherent Database Representations (CDRs) which are used in the query validation and decomposition process. Changes are applied as directed by the IDNSF.

The IDNSF must distribute to the D2Q Administrators all changes which are to be applied to the D2Q dictionaries; these dictionaries are the same at all D2Q nodes. In addition, the IDNSF must assure that all D2Q nodes have received the changes and must coordinate the application of

the changes to the D2Q dictionaries. This coordination must assure a smooth cutover on query processing, so that no query fails as a result of applying dictionary changes. Although it has been stated that the D2Q node administrator should apply the changes at a given D2Q node, the procedure should be as automated as possible; e.g., via a trigger from the IDNSF.

4.2.2.3 The D1* Processor

The D1* processor's primary functions are as follows:

- It translates the network access names in subqueries to their corresponding names in the the local database.
- It ships the resultant subquery to the data node for execution.
- It receives the subresponse from the data node.
- It operates on subresponses which are input to other subqueries.
- It transmits subquery status information to the controlling D2Q.
- It transmits the subresponse to the target D2Qs.

The D1* Administrator's role centers on making the data resident at a Data Node available to the network. In particular, the administrator must perform the following tasks:

- The D1* administrator must identify the subschema of the target database which is appropriate for the network.

- If the target database is new to the IDN, the Dl* Administrator must obtain a CDR for it from the IDNSF. Thus, a CDR-ELEMENT and CDR-RECORD-TYPE must be identified for each corresponding data entity in the subschema which is accessible to IDN queries. All of this information must be input into the Dl* dictionary.

- As changes are made to the IDN subschema at the Data Node, the Dl* Administrator must notify the IDNSF of the changes, and of any potential changes to the Dl* dictionary.

In addition to the basic facilities for defining and maintaining CDRs and their correspondences to target databases in the Dl* dictionaries, the IDNSF must provide facilities to:

- Integrate the CDR data definitions with those of other CDRs. In particular, the proper identification and use of CDR-ELEMENTS must be supported by automated tools. A recurring problem situation which must be supported by tools is the ability to identify when a given data element in a target database corresponds to an existing CDR-ELEMENT, and when a new CDR-ELEMENT is required. The IDNSF should provide facilities to assist in making this decision in an objective manner.

- Develop a normalized CDR for a given database subschema.

- Distribute new and modified CDRs throughout the network.

4.2.3 Network Administration Requirements

Network administration is charged with the following responsibilities:

- Planning modifications to the network.

- Making changes effective in the network.

The IDNSF must support Network Administration by providing the following capabilities:

- Network configuration decision support.

  A model of the current IDN configuration should be maintained. The IDNSF should provide a tool whereby modifications to the network configuration could be specified, and/or changes in loads could be specified. This tool could then determine the potential effects of the proposed changes on network performance.

- Making changes effective in the network.

  Whenever a node is added to or dropped from the network, each of the nodes must be notified not only of the change, but also when the change is to become effective. Similarly, if a given node's assignment (in terms of user community or target database or backup for another node) is changed, this change must be sent to affected nodes.

  It must be assumed that it is not possible to propagate this information instantaneously. Since the network must operate without stopping, it is reasonable to expect that there will be a short time with parallel configurations, i.e., when old queries continue to process as if the change were not in effect, but new queries process against the new configuration.

4.3 THE IDNSF

The IDNSF makes available to the IDN Support Group the aggregate of all tools required by this Group to allow it to fulfil its role in supporting IDN users and Operations Management.

These tools may be categorized as follows:

- A software development environment.
- A data analyst "workbench".
- A configuration management system.
- A distribution & distribution control system.
- An IDN simulation.
- A service request reporting system.
- A time and project reporting system.

These tools are complementary, and it is desirable that they should be as integrated as possible. In the discussions which follow, tool interfaces and integration are discussed. The degree of integration of these tools will, however, be determined by many practical considerations. These considerations include but are not limited to:

- The organization and geographic distribution of the IDN Support Group.
- The IDN processor hardware/software architecture.
- The variety of hardware/software configurations at User Nodes and Data Nodes.
- The hardware/software configuration used for the development environment, which may differ from that for both the IDN processors and the user and data nodes.

- Existing procedures and systems which fulfill some of the functions identified above.

4.3.1 The IDN Software Development Environment

The IDN Support Group must have adequate facilities for software development. This implies the existence of basic development facilities, such as:

- Hardware to be used for development.
- Compilers.
- Linker/Loaders.
- File Systems.
- Editors.
- Debugging Facilities.
- A System Library.

It is not necessary that the development hardware be the same as the operational hardware. Indeed, to the extent that the IDN processors are customized to support their role in the IDN, there is an increased probability that development hardware is different from the operational hardware. It is, however, expected that operational hardware will also be available at the IDNSF for operational testing.

Ideally the Configuration Management System should be integrated into the development environment. This integration would serve several purposes:

- It ensures that every program and/or file which is to packaged in a release of the IDN software is known. No program or file will be inadvertently dropped from the software release. Conversely, it also ensures that only required components will be included in a software release.

- It ensures an exact correspondence between source and executable objects. Not only does the lack of such a correspondence present difficulties to people who must analyze problems, it is mandated by TCB security requirements.

- It can be used to identify differences between different releases of the IDN software. Knowledge of such differences can be helpful in problem analysis.

- It can be used to relate specific versions of programs or modules to problems and/or requirements which they address.

The Configuration Management System is considered to be based upon the dpANS IRDS, which provides a basis for addressing the above problems.

It is assumed that there will be an interface between the development environment and the IDN simulation. This interface would consist of the utilities to unload software from the development environment and load that software on the simulation. This would permit:

- Testing the upward compatibility of new software releases.

- Volume / performance / stress tests of new software releases.

- Running software with special diagnostics on replications of field problems.

4.3.2 The Data Analyst "Workbench"

The IDN Support Group has the responsibility for creating and distributing "global metadata" across the IDN. This global metadata includes:

- CDRs for each of databases accessed by the IDN.
- Mappings from each CDR to the corresponding IDN-Database-View.
- Sets of CDR-ELEMENTs which are used in establishing and maintaining the LUNS for each user community using the IDN.

Centralizing this function is necessary because the CDRs constititute the IDN's global view of the data available. It is imperative that the CDRs and their mappings be properly defined. If they are not, queries will either not process, or worse, process erroneously.

Central to the Data Analyst Workbench is a dictionary which contains:

- An inventory of Data Nodes.
- The corresponding IDN-Database-Views for those Data-Nodes.
- An inventory of User Nodes and the security-classifications of users at those nodes. (The profiles of individual users or groups are not known the IDN Support Group.)
- All CDRs which are active in the IDN.
- The definition of all IDN Nodes and their responsibilities for supporting User and Data Nodes and their shadowing responsibilities.

The Data Analyst Workbench is a collection of IRDS-based tools which facilitate the development and distribution of CDRs and their mappings to IDN-Database-Views to the appropriate nodes in the IDN.

In addition to the normal maintenance and reporting facilities of the IRDS, the following special facilities are included:

- Normalization Utility.

  This utility assists a data analyst in developing a CDR which is assured to be in 3rd Normal Form. Query processing assumes that the CDRs are in 3rd Normal Form.

- Corresponding / Duplicate Element Finder.

  If an IDN query is to join data from two or more databases, the same data element(s) must be common to the CDRs for the different databases. Thus it is important to identify likely inadvertent duplication of data elements (e.g., if BIRTH-DATE and DATE-OF-BIRTH are both CDR-ELEMENTS, they are probably duplicates.) Likewise, given an IDN database's subschema, it is desirable to identify the CDR-ELEMENT which corresponds to a database data-element should it exist. (If DATE-OF-BIRTH were the element as defined in the schema of the target database, and BIRTH-DATE were already defined as a CDR-ELEMENT, probably the database data-element DATE-OF-BIRTH should be mapped to the CDR-ELEMENT, BIRTH-DATE. This CDR-ELEMENT would then be used in defining the CDR for the given IDN database.)

- DPP Consistency Analysis.

  The DPP for a given CDR-ELEMENT designates which database should be accessed when certain values of that CDR-ELEMENT are requested. Given the possible overlap in values across databases, it is desirable to ensure that the DPP for a given CDR-ELEMENT is consistent with its domain in a given target database. A more subtle problem occurs in defining the DPP consistently for across several CDR-ELEMENTs. (It might be pos-

sible to define DPP for CDR-ELEMENTs A and B such that no combination of A and B would ever be retrieved.)

o Dictionary Import/Export.

This utility is a native IRDS facility, but is mentioned here because of its special importance to this support function. With it, it is possible for the IDN support group to collect IDN-Database-View descriptors from the various Dl* Nodes so that their CDRs and mappings may be defined. It also provides the mechanism for global metadata distribution. Finally, it also provides an interface to the IDN simulation. Given the importance of correct CDR and mapping definitions, it is desirable that these be tested in advance of their distribution.

o NANS Usage Analysis.

This utility finds all CDR-ELEMENTs which are accessible by users with a given security classification. With this utility, it is possible to identify and/or create those subsets of the NANS which are required for the construction of LUNSs for a given User Node. By determining this prior to the distribution of CDR-ELEMENTs to Dl Nodes, two purposes are served:

- The LUNSs defined at the Dl Nodes are assured to be correct. If a given user's LUNS contains a LOCAL-ELEMENT-NAME which is associated with a CDR-ELEMENT which is inaccessible according to the user's security classification, any query using that LOCAL-ELEMENT-NAME will fail with a security violation. This approach can prevent this undesirable situation from occurring.

- It facilitates the distribution of CDR-ELEMENTs to Dl Nodes on a as-needed basis. If a CDR-ELEMENT is not

needed by any user community supported by a given DI node, it will not be sent to that node.

This utility interfaces with the import/export utility to create the NANS subset for a given User Node.

Since both the Configuration Management System and the Data Analyst Workbench are both based on the IRDS, integration is automatically provided. As seen earlier, incorrect metadata in the IDN dictionaries can cause processing errors, in a manner analogous to software "bugs". Just as in the case of software, the metadata must go through a controlled evolution, and the impact of its changes must also be tested in advance.

4.3.3 The IDN Configuration Management System

The IDN Configuration Management System consists of the IRDS, special interfaces to the development environment, and procedures. It provides for:

- Controlled development of all IDN software.
- Controlled development of global metadata.
- A methodology which assures an orderly evolution of both software and metadata.
- Quality assurance for both software and metadata.
- TCB provability for the IDN software as changes are made.

The following basic functionality is provided by the IRDS:

- o Baseline identification and control.

  This implies the ability to prescribe the expected level of completion of objects (configuration items) for each particular baseline; e.g., if the baseline is for software specifications, it is reasonable to expect that:

  - All relevant (e.g., those belonging to a particular subsystem) programs and modules be identified.
  - That the required functionality for each be specified.
  - That interfaces among them be specified in terms of the data being passed between them.
  - That the logical structure of all related databases, input screens and output be specified.

- o Configuration Item Change Control

  This implies that once an object passes (forward) through a baseline, changes to that object are restricted to only those characteristics which were not required in order to initially move through the baseline. If one of these particular characteristics requires change, then the object must be placed "before" (i.e., moved back through) the baseline and must be revalidated before allowing it to move forward again through the baseline. This is necessary to assure that the integrity of the baseline is preserved.

- o Impact of Change Analysis

  This functionality is necessary to allow a "Configuration Control Board" to know what the potential effect of a change

might be, at least in terms of the magnitude of the number of objects (configuration items) effected by the change. It is also of obvious use in developing cost/benefits tradeoffs, in order to determine a proposed change should be made.

It is important to distinguish between the concepts of configuration control and configuration management. Configuration management is the set of procedures (manual and automated) for establishing (and disestablishing) baselines, for deciding when and what changes should be made, and for making changes to configuration items which are part of a baseline. Configuration control is the set of procedures (manual and automated) which prevent changes to a configuration, except under change control. It should be obvious that configuration control is a necessary feature of configuration management.

4.3.4 The IDN Distribution & Distribution Control System

This system performs the following functions:

- It creates distribution copies of both software and dictionary descriptors to be shipped to the various nodes. It is reasonable for each specific version of the software to be archived at this point.

- It maintains an inventory of the software and dictionary descriptor releases sent to each node.

- It tracks the releases of software which are active at each node. This information can be very useful for the analysis of problems.

4.3.5 The IDN Simulation

The IDN Simulation is a test model of the IDN. It may contain a miniature network, or the network may be simulated within a single machine. The simulation should contain both a set of sample test databases and dictionaries. It also should contain dictionaries which can support access to any database in the IDN. It also should contain databases with dummy data. In this way, the IDN Support Group could simulate problem situations without having access to secured data.

4.3.6 The IDN Service Request Reporting System

This system is used to capture requests for service from the IDN Support Group. The types of service may include:

- Requests to correct operational problems.
- Requests for enhancements.
- Requests for new copies of software and dictionaries.
- Requests for information.

These requests should be captured and categorized, so that a profile of the types of service may be obtained. For requests which do not involve enhancements, the time and personnel resources required to provide the service should be captured. Each request for service should also identify both the requestor and time requested. Reports can then identify both "problem requestors" and old open requests.

This system should interface to both the IDN Support Group Activity Reporting System and the IDN Configuration Management System. It is not be desirable to attempt to directly relate software changes to service requests. A request for either a software correction or enhancement may translate into many technical requirements.

4.3.7 The IDN Support Group Activity Reporting System

This system is used for both project management and for analysis of time and resources required to service requests. Although the resources required to correct problems and provide information are unpredictable, it is necessary to identify how many resources are required and to identify trends. If a serious problem interferes with project activity, this can be identified.

4.4 OPERATIONAL SUPPORT UTILITIES

The IDN must have a set of utilities and procedures which assist in managing the day to day operation of the IDN. These utilities and procedures perform various services. In the following section, utilities will be identified for each class of service. Procedures, both manual and semi-automatic, will then be discussed.

The table given on the following page summarizes the types of utilities provided by the Operational Support Facility and classifies them according to the service provided. The table also shows which type of administrator requires the particular utility.

Class of Service	D1	D2Q	D1*	IDN
Node Operation	*	*	*	
Query User Support	*			
Monitoring & Analysis	*	*	*	*
Modeling & Design Aids			*	*
Change Implementation	*	*	*	*
IDNSF Interface	*	*	*	

4.4.1 Node Operation Utilities

Although it is anticipated that IDN nodes will operate automatically, several simple functions which require operator intervention are needed:

- Node Start

  This utility will load the IDN software and establish the necessary connections to the IDN. This involves broadcasting a message notifying the network that the given node is available for processing.

- Node Restart

  This utility restarts a node after a failure or shutdown. In addition to performing the normal functions of Node Start, this utility will synchronize the node's metadata with its shadow nodes and those nodes for which it acts as a shadow.

- Node Shutdown

  This utility performs a graceful shutdown of a given node. This probably involves notifying the IDN that the node will accept no new work, and completing the current work assigned to it. If the node is unable to complete some work within the grace period, the work units which have not been completed must be reassigned to another node.

- Node Tuning Utilities

  It is anticipated that each IDN Node may have several software "switches and dials" to allow the administrator to optimize the node's performance. On the other hand, it is possible that the IDN nodes could be made to be entirely self-monitoring and self-adjusting. In the latter case, these utilities would not exist.

4.4.2 Query-User Support Utilities

The utilities that support the query user fall into two categories:

- Local User Name Space (LUNS) Maintenance Utility.
- Query Execution Control Utilities.

The LUNS Maintenance utilities are to assist a DI administrator in establishing or modifying a LUNS. As noted in Chapter 3 (Subsystems

Specification), if the D1 administrator were to rely on dictionary maintenance alone to create or modify a LUNS, it is possible that one or more LOCAL-USER-NAMEs in the LUNS could be related to a CDR-ELEMENT which the user is not authorized to use in a query. Since a given D1 dictionary may support several different users with very distinct profiles, the corresponding dictionary may contain some CDR-ELEMENTs which are not visible to all query users. The LUNS Maintenance Utility would provide the means whereby:

- A default LUNS could be created for any given query user.

  The default LOCAL-ELEMENT-NAME associated with a given CDR-ELEMENT could be created from a descriptive attribute from the CDR-ELEMENT. The administrator could then use the IRDS Modify-Entity-Access-Name Command, or a specialized dialogue which invokes this command, to provide the name appropriate for the particular use.

  In those cases where a new version of an existing CDR-ELEMENT is imported, the LOCAL-ELEMENT-NAME could be related to the new CDR-ELEMENT and disassociated from the old CDR-ELEMENT.

- CDR-ELEMENTs which are no longer visible to a given query user are identified, and optionally the corresponding LOCAL-ELEMENT-NAME are deleted. Stored queries which reference the obsolete LOCAL-ELEMENT-NAMEs are identified and marked as invalid.

This use of this utility not only avoids problems, but it also eases the work of the D1 Administrator.

The Query Execution Control Utilities are used to override the normal processing of queries on an exception basis. The following utilities are anticipated.

- Cancel Query
- Hold Query / Output
- Restart Query
- Send Output
- Query Status
- Change Query Priority

4.4.3 Monitoring and Analysis Utilities

The monitoring utilities gather statistics at each node and generate summary data for given time-frames.

The monitoring utilities are always active whenever the IDN is active. They fall into three categories:

- Detecting and Identifying Failures.
- Issuing Warnings.
- Collecting and Analyzing Statistics.

Both failures and warnings require that those with the ability and authority to take corrective action be notified. The difference between the two categories is that failure detection and identification occurs when a problem has occurred, whereas warnings are issued whenever the potential for a problem exists. Thus warnings are important for problem avoidance.

The design of failure detection and warning utilities must take into account that, to a large measure, each IDN processor should function

without an operator. Given this, the party which takes corrective action in most cases may be another utility. The failure detection and warning utilities also must interface to the statistics utilities, since the occurrences of failures or warning states are important statistics.

Statistics are collected to:

- Determine IDN performance characteristics.
- Plot trends and aid in capacity planning.
- Assist in tuning the configuration of IDN processors.
- Assist in configuring and tuning IDN processors.

Statistics are gathered at each IDN node, but the type of statistics obtained will vary according to the type of the IDN Node. In addition, the individual node statistics must be summarized to provide an accurate statistical picture of the network. The types of network statistics should correspond to elements in the network performance model. In this way the predictions given by an IDN performance model may be checked against actual performance data.

Since statistics are computed over varying time frames, it is necessary to provide adequate on-line secondary storage. In order to analyze long-term trends, these statistics must be archived. This implies that off-line secondary storage should be available.

Finally the existence of a database of IDN statistics implies a querying and reporting facility. It may be assumed that there are a number of standard or "canned" reports which could be produced cyclically. These include but are not necessarily restricted to:

- Summary of Activity
- Trends Report

- Peak Load Analysis.

The statistics reporting facility should provide for graphical presentation of statistical data. It is necessary to identify both the proper technology for producing such reports (e.g., the use of PC-based tools or mainframe graphics packages), and the frequency of each of the reports.

Additionally, there is the need to produce ad hoc statistic reports. It is likely that the standard report formats will suffice for most ad hoc reports. In this case, the ad hoc reports would be produced by varying the selection criteria for standard reports. The requirements for graphic presentation also apply to ad hoc reports.

The mix of cyclic and ad hoc statistical reports must be identified. Finally, it must be decided whether these reports should be retained for historical purposes. If so, this could greatly increase the secondary storage requirements.

4.4.4 Modeling and Design Aids

There is a need for modeling and design aids in two areas:

- Network Performance Modeling.
- Data Design.

The Network Performance Modeling utility enables a network administrator to determine the performance characteristics of the IDN given a specific network configuration and workload. If possible, existing software should be used to develop and maintain the IDN Performance Model. There should be an interface between the IDN dictionary (each of which contains a description of the existing network configuration) and the statistics collection utility (which identifies a workload for a certain

period) and this modeling utility. A network administrator then could easily specify a new network configuration as modifications to the existing configuration: vary the workload against the new configuration, and compare the behavior of the proposed configuration to the behavior of the existing network.

The IDN can process user queries only if the data structures of the target databases are properly defined to the DI* nodes and properly mapped to Coherent Database Representations (CDRs). To help ensure that the data structures are cleanly and correctly defined, the following utilities are envisioned:

- Normalization Utility.

  Normalization of data structures is a valuable exercise even in developing a non-relational database schema. The closer the target database is to 3rd Normal Form, the simpler will be the mappings from the CDR to the IDN-Database-View for that database.

  This is the same utility used by the IDN Support Group. This tool could aid the database designer in those cases where the DBMS does not provide such functionality. In these cases, specialized interfaces would be required to transform the resultant schema in the dictionary used by the normalization utility into the database schema used by the target DBMS.

- Corresponding / Duplicate Element Finder

  This is a modified form of the utility used by the IDN Support Group, and is intended as a database design aid. The problem of duplicate element definition applies to each database schema as well as in the CDRs and their corresponding IDN-Database-Views. As is the case with the normalization utility, this utility is provided to Data Node Administrators

for those cases where the corresponding functionality is not provided by the target DBMS.

- Database-View Generation / Load.

  Each Dl* dictionary contains an IDN-Database-View for the target database. This database-view corresponds to a sub-schema of the target database for the IDN. In some cases, this view may be the same as the database schema. Ideally, this subschema should be defined only once - probably at the Data Node. A utility should transform this subschema into Dl* dictionary descriptors. These descriptors are then sent to the IDN Support Group, which will create the corresponding CDR and mapping relationships.

  For those databases with a stable schema and/or an uncommon DBMS, manual input if the (sub)schema into a IDN dictionary may be warranted.

4.4.5 Change Implementation

The IDN will evolve over time. This evolution requires that utilities, controls and procedures be in place to ensure that changes in the IDN are implemented smoothly. The changes fall into three categories:

- Database Changes.
- Network Changes.
- Software Changes.

All three categories share certain requirements. They are:

- Continuous Operation.

The entire IDN must continue to operate without disruption.

- Synchronized Transition.

  As warranted, changes must take effect simultaneously at all affected nodes in the IDN.

- Graceful Transition.

  The implementation of a change must not affect queries which are executing.

These requirements dictate that a common approach be devised to implementing each of the types of changes cited above. One such approach is presented after the details of each of those types of changes is discussed.

4.4.5.1 Database Changes

Database changes fall into two categories:

- Switching Physical Databases.
- Switching both Logical and Physical Databases.

4.4.5.1.1 Switching Physical Databases

The change of physical database means that for a period of time there are two databases at the Data Node, and "old" database and a "new" database. The new database may either contain new data, or it may be a physical reorganization of the old database. In the latter case, the content and logical structure are identical, and the differences are to enhance performance. The new database may have space allocated differently, or data partitioned across physical devices differently, or

some other differences in physical structure. (It is assumed that each DBMS at the Data Nodes provide utilities for reorganizing, unloading, reloading, and activating the database.) After the physical the new database is created, there is a need to gracefully switch IDN queries from processing the old database to the new database. Although this type of database change is the easiest to accomodate, it may be performed frequently.

A Database Switch utility can provide the required graceful cutover. The utility consists of a filter which captures all IDN query requests against a database. When queries are to be redirected to a new physical database, the data administrator at the data node informs the IDN that at a given time, all queries are to be directed to the new physical database. The filter continues to direct all subqueries for queries initiated prior to cutover against the old physical database. When all those queries are processed, the administrator is notified that the old database has no activity. Upon receipt of this message, the administrator may delete the old physical database.

4.4.5.1.2 Switching Both Logical and Physical Databases

A target database normally evolves in response to changing local data requirements. At least some of the changes in the target database are deemed to be of interest to IDN users.

Changes in the logical structure of any database have a wider impact on the IDN than merely changing physical databases. Any change to an IDN-Database-View will at least affect the mapping relationships between the CDR and the IDN-Database-View. More often, such modifications to the IDN-Database-View of a database will also generate changes such as: adding new data-elements; changing of keys, and/or defining different functional dependencies. Each of these imply changes in the corresponding CDR (and potentially DPP.)

The following actions are required to implement changes to IDN-Database-Views:

- Changes to the database's corresponding IDN-Database-View must be specified, and a new IDN-Database-View must be prepared for the target database.

- Each new IDN-Database-View must be shipped to the the IDN Support Group. The planned effective date for the IDN-Database-View must also be identified.

- The IDN Support Group must perform the following tasks:

  - The impact on the CDR and DPP must be analyzed.

  - Corresponding changes to the CDRs and DPP relationships must be specified and approved.

  - The mapping between CDRs and IDN-Database-Views must be specified and validated.

  - The new dictionary descriptors must be distributed to all D2Q Nodes and affected D1* nodes. These new descriptors include:

    * New or modified CDRs.

    * New or modified IDN-Database-Views.

    * New or modified CDR-ELEMENTs, VALUE-SETS, and DPP relationships.

  - Dictionary descriptors must be prepared for distribution, and identifies the cutover point for implementation across the network.

- If one or more CDRs are affected, new dictionary descriptors must be distributed to each D2Q node.

- All new or modified CDR-ELEMENTs must be distributed to those D1 dictionaries which are affected. All CDR-ELEMENTs which are obsolete, or have had changes in security classification must be identified, since these changes may invalidate currently operational user queries.

- The new IDN-Database-View, CDR, and mapping relationships must be distributed to each each D1* dictionary which supports the Data Node with the corresponding IDN-Database-View.

o At the Data Node the following actions must be taken:

  - The target database must be converted to the new logical structure.

  - New data must be loaded into the new database as required.

o At affected D1 Nodes, new CDR-ELEMENTs must be incorporated in different LUNSs as appropriate. (It is conceivable that some CDR-ELEMENTs may be deleted from one or more LUNSs. In those cases, the users should be warned and their affected queries be identified.)

o The modified LUNSs, CDRs, IDN-Database-Views and corresponding target databases must be activated simultaneously across the network. At the cutover point, in-flight queries must continue to process until completion, while new queries process using the new CDRs and databases.

The following utilities are required by operations management to implement this class of database changes:

- Dictionary Import/Export.

  This is IRDS standard functionality. It is used to support the following tasks:

  - Exporting IDN-Database-Views to the IDN Support Group.
  - Importing dictionary descriptors from the IDN Support Group, and creating a new dictionary with them or loading them into an existing dictionary.

- Switch Utilities.

  A dictionary switch utility performs the switching of query processing from one set of dictionary descriptors to another at the designated time. Additionally, the Database Switch utility identified in 4.4.5.1.1 also must be used here.

- Database Utilities.

  It is assumed that each DBMS at the Data Nodes provide utilities for reorganizing, unloading, reloading, and activating the database.

4.4.5.2 Changing the Network

There are four basic categories of network changes to be considered:

- Adding a new node.
- Removing a node from the network.
- Changing the assignment of shadow nodes.
- Changing a node's responsibilities.

Adding a node to the IDN or removing a node implies much more than the simple activation or deactivation. Each node may act as a shadow for other nodes. Similarly, each node may be shadowed by other nodes. Thus node additions and deletions will affect the responsibilities of other nodes.

Whenever a node is to be assigned as a shadow for another node, the shadow node's dictionary must contain all the descriptors in the primary node's dictionary for its primary responsibility.

> Note: because of the variety of ways shadow assignments can be made, it is not necessary that the dictionary contents of D1 and D1* nodes be identical. Suppose there are three nodes, A, B, and C. Further suppose that B shadows A, C shadows B, and A shadows C. Then:
>
> - Node A's dictionary must contain the descriptors necessary to service queries by both A and C.
> - Likewise, B's dictionary must contain descriptors to service A and B queries.
> - Finally, node C's dictionary must contain descriptors for B and C's queries.

The import/export and internode dictionary verification utilities identified earlier are sufficient for synchronizing the dictionaries.

When the network is to be reconfigured, a cutover problem similar to the one encountered for databases occurs. In this case, it is desirable that all nodes gracefully switch to processing according to the new network configuration based on the time of origin of query transactions. This implies that network definition descriptors be stamped with an "as-of" date/time-stamp. This allows for preloading descriptors, which can facilitate a synchronized, graceful switchover to the new configuration.

4.4.5.3 Software Installation

The installation of new releases of IDN software will require a synchronized switchover only in those cases where message formats are modified. Although this situation may be rare, anticipation of this occurrence can eliminate the problem. The same approach for achieving a synchronized, graceful switchover as described above for network configurations can apply to converting to a new release of the IDN software.

4.4.5.4 The General Approach to Change Implementation

Recall the three basic requirements to implementing the changes discussed above. They are:

- Continuous Operation.
- Synchronized Transition.
- Graceful Transition.

A simple and effective way to achieve these objectives is through a standard message filter within the network input subsystem at each IDN node. Under this scheme, a "transition-event table" is maintained for each node. For each unique as-of-date/time stamp, a list of run-time parameters is specified. These run-time parameters should identify the software library; the dictionary, and (in the case of data nodes) the physical database files to be used. The parameters may be given finer granularity. For example, individual versions of CDRs, LUNSs, and Database Views might be identified instead of a different dictionary.

Whenever a message is processed,

- It is identified with an as-of date/time stamp. (For most messages, this is the date/time stamp for the origin of the associated query.)

- The message's as-of date/time stamp is matched against the transition-event table. The table entry whose key is closest to but no greater than the message's date-time stamp identifies the run-time parameters for the message.

- The filter then activates a task which uses the software, dictionary, and database(s) identified in the corresponding table entry.

This approach permits messages with different as-of date/time stamps to process with different run-time parameters. As old queries complete processing, the corresponding messages will be purged from the IDN. Eventually all queries will process with the new run-time parameters. Where multiple nodes are involved in a given transition, the transition-event table at each node has an entry for a given date/time value. Thus the transition from one set of run-time parameters to another is graceful, instantaneous, synchronized across multiple IDN nodes, and provides for continuous operation of the IDN.

4.4.6 IDNSF Interface

Operations Management has six major needs for communication with the IDN Support Group:

- Network reconfiguration.

- Implementing logical database changes in the IDN.

- Implementing IDN software changes.

- Resolving operational problems.

- Requesting software enhancements.
- Requesting some form of technical assistance.

IDN users share the last three requirements with Operations Management. Each of these interactions with the IDN Support Group is discussed individually.

4.4.6.1 Network Reconfiguration

There are two cases to be considered:

- Establishing a new node.
- Reassigning node responsibilities.

In the former case, the IDN Support Group must be notified in advance of the node's requirements. This is a normal type of service request, and their should be standard forms for this type of service request. If the node to be established is a Data Node, then the supporting D1* nodes must be identified, and the IDN-Database-View for each database at the Data Node must be provided. In the case of a User Node, the supporting D1 Nodes must be identified, and the security classifications of users at the User Node must also be provided. The IDN Support Group must then update its distribution profiles, and create and ship both the most current release of the IDN Software and the appropriate metadata.

In the case of reassigning node responsibilities, it is possible for the transfer of metadata to be accomplished entirely by operational personnel. In complicated situations, the IDN Support Group should be contacted for technical assistance. It is necessary that the IDN Support Group be notified of pending changes, so that the simulation model can be updated. This will facilitate the IDN Support Group's ability to respond as required.

Since each node of the IDN carries the entire static network view, it is possible to have the transfer of this metadata to the simulation model occur automatically. This could be accomplished entirely by a utility within the IDNSF, provided that the IDN Support Group has an IDN User Node. If not, the IDN Support Group would require some formal notification of network changes from IDN Network Management.

4.4.6.2 Implementing Logical Database Changes

The details of this type change were discussed in 4.4.5.2. It is clear that the IDN Support Group will require a formal service request and adequate lead time to provide this type of service. Procedures must be in place so that Node Administrators must acknowledge receipt of the new dictionary descriptors, and that the IDN Support Group is notified when these descriptors are applied to the node dictionaries. Potentially this notification could be generated automatically. However, in the case of D1 Nodes, the D1 administrator must still perform additional maintenance to establish or modify LUNSs.

4.4.6.3 Implementing IDN Software Changes

It is assumed that all software changes shall be applied as directed by the IDN Support Group. The transition from old to new software will be accomplished by a switch as previously discussed. The major requirement here is that adequate lead time must be given to Node Administrators so for loading the new software. Procedures must be in place so that Node Administrators must acknowledge receipt of the new software, and that the IDN Support Group is notified when the software is loaded. It is possible that software installation procedures could automatically send a message to the IDN Support Group after software is loaded and ready.

4.4.6.4 Resolving Operational Problems

Large system vendors have three common problem in determining the source of user problems:

- Determining the hardware/software/metadata configuration in use.
- Recreating the conditions which caused the failure.
- Resolving the problem in the shortest possible time and without introducing new problems.

All of these problems will confront those responsible for maintenance and enhancement of the IDN. In addition, the IDN Support Group must make changes in strict conformance to configuration management procedures, so that the security of the IDN or its supporting utilities is not compromised.

The first two problems can be addressed by procedures and utilities which interface to the IDN Distribution Control System. Note that the each IDN node must be aware of its node-type and of the software which is operating at the node. Furthermore, the IDN itself is aware of all the nodes in the network. Thus it is possible to establish an interface whereby the IDN support group can know the exact hardware/software configuration at each IDN Node. The IDN software installation procedures can ensure that the IDN support group receives this information. In the case where custom software is required for a given User Node or Data Node, manual procedures will be required to ensure that the IDN support group has at least source code for the custom software.

In order to resolve problems, two strategies are commonly employed:

- Debug the problem at the problem node(s).
- Recreate and debug the problem at the support site.

Debugging problems at the nodes where they occur usually involves analyzing actual files and memory dumps where operational data can be seen. Normally, the people in the IDN Support Group responsible for software maintenance or metadata management do not have access to operational data. Thus such techniques for on-site debugging are excluded from consideration.

(Presumably, such on-site debugging could be employed if there are individuals in the IDN Support Group with both the technical competence to analyze the problem and the appropriate security clearance to view the operational data present when the problem occurs. However, depending on the nature of the problem, the existence of such an individual may not be able to be determined.)

The IDN Simulation discussed in 4.3 provides the basic mechanisms for recreating problems. The IDN Support Group's basic requirement is then obtaining enough information to recreate the conditions which caused the problem.

It is presumed that the IDN Support Group may receive a copy of any query which was running when the IDN failed. However, it may not see the actual data which was returned by the query.

Dictionary Import/Export may be used to provide the IDN Support Group with actual queries. However, dummy data must be used to simulate the problem. Naturally, there is no assurance that the problem can be recreated if dummy data is used.

Since a problem may not be reproducible in the IDN Simulation, it may be necessary to design diagnostics into the IDN software. Such diagnostics could reveal much about what is happening when the problem occurs, without revealing any actual data. Since the diagnostic code is part of the IDN software, it must conform to the same TCB security criteria as the code which actually performs the query processing. Thus the output of such "sanitized diagnostics" would not present a security

problem. It does, however, impact the performance of the IDN. In the case where there is a timing-related problem, the execution of diagnostic code itself might change conditions sufficiently so that the problem could not be reproduced.

Given the complexity of the IDN software, and the amount of processing and data manipulation at each node, it is not sufficient to have a single diagnostics on/off switch. Diagnostics must be able to be turned on and off selectively in the IDN software. Additionally, it may be necessary to provide facilities to stop and restart the IDN software at various critical points. Stops must be able to be selectively triggered by events (e.g., the receipt of a subresponse from a given node.)

Once a problem has been analyzed, the proposed correction must be assured as not violating TCB security criteria. This implies that all IDN software changes go through the same stringent procedures that the initial system went through.

4.4.6.5 Requesting Software Enhancements

All r quests for Software Enhancements from the field should be submitted formally to the IDN Support Group. The enhancement request should be accompanied by appropriate documentation which corroborates the need. The request should also designate one or more individuals to contact so that the IDN Support Group may obtain any such clarification that it requires.

4.4.6.6 Requesting Technical Assistance

Requests for technical assistance may be formal (i.e, in writing) or informal (via telephone.) Certain categories of technical assistance will require formal requests. The IDN Support Group should publish its policies on this matter.

4.4.7 Procedures

The implementation of various types of changes in the IDN will require the coordination and cooperation of people with different responsibilities. In particular, procedures and guidelines are required for the following:

- Requesting IDN Support Group Services

  Formal procedures for requesting the technical assistance of the IDN Support Group are required for:

  - Obtaining subsets of the NANS to construct one or more LUNSs.
  - Implementing database changes which change the corresponding IDN-Database-View.
  - Obtaining software and metadata for establishing a node.
  - Assistance in reassigning node responsibilities.

  Additionally formal procedures are required for:

  - Problem reporting.
  - Requesting enhancements.

  Finally, error correction and analysis guidelines should be established to ensure that operational problems which can be solved locally are handled locally.

- IDN-Database-View / CDR Changes

  Whenever the logical structure of any given database is changed, the changes to its DATABASE-VIEW and mapping relationships to the CDRs must be distributed to all D1* dictionaries which support the database. Shadow databases must also be converted to the new structure. The cutover point must be identified, and specified at each affected processor.

  If the CDR is affected, the impact is even wider. Any new elements in the database must be identified. They must be analyzed to determine if they duplicate the elements in other databases. If so, the impact on DPP must be determined. New CDRs, DPP definitions, and mappings to the DATABASE-VIEW must be defined and distributed. The changes in LUNSs must also be implemented. A switchover time must be designated and specified at each affected IDN node.

- Network Reconfiguration

  The dictionaries at each of the affected nodes must be changed. A switchover time must be designated and specified at each affected IDN node. Any new nodes must be activated prior to transition to the new network configuration. After transition has been completed, old nodes must be deactivated, and dictionaries must be purged of descriptors which support processing assigned to other nodes.

- Software / Hardware Upgrades

  The IDN support group must be informed of all software / hardware changes and the switchover time.

4.5 ISSUES

This section presents a discussion of several outstanding issues for the IDNSF and the Operational Support Utilities.

4.5.1 IDNSF Issues

1. Support for global metadata management is quite independent of support for the IDN software. The former requires people with experience in data analysis and data base applications; the latter requires software engineering skills. Thus, it is possible to separate the IDN Support Group according to these two functions. This would imply, however, that distribution control over software and metadata are handled separately. This will affect the procedures which IDN Users and Administrators request support services.

2. This document has assumed that the IDN Support Group has control over the software installed at each of the IDN processors, but does not have control over software in use at either the User Nodes or Data Nodes.

   The assumption regarding the IDN Software has two major effects:

   - It reduces the complexity of the support problem, since (except for transition periods) only one version of the software is in use at any one time.

   - It simplifies analysis and verification of TCB criteria.

3. The data analysis utilities will make use of the IRDS implementation used for the IDN processors. If the dictionary systems which support the individual DBMSs conform to the IRDS standard, and if their schemas are compatible with the one used in the IDN, then the

problem of communicating between the dictionaries would be eased. If the dictionary systems at the Data Nodes are different from that used by the IDN, then the use of the data analysis utilities would be limited unless either they could be customized at the Data Nodes, or that special inter-dictionary interface software be written.

4.5.2 Operational Support Utilities Issues

1. A user has the capability to inquire about a query's status, cancel a query, hold its output, or send its output.

   A more sensitive question arises regarding a user's capability to change the priority of a query (particularly if a higher priority is requested). This is an area where a policy or set of policies will have to be established. Associated with this issue is the question of what kind of monitoring should such an individual should have. Since IDN is a large network, the cost of obtaining network level status information itself could be high.

2. The Physical Database Switch Utility could operate at either the D1* or Data Node. If the utility is resident at the D1*, then the utility would also have to communicate with all D1* Nodes which support a given Data Node. This introduces some additional complexity and IDN messages. However, this software would require only a single implementation. If the utility resided at the Data Node, then the issue is very simple and the IDN would be completely unaware of this type of change. On the other hand, differences in Data Node hardware and operating systems would require different implementations of this utility. The question of IDN Support Group responsibility for and distribution of the various implementations becomes an issue.

3. There is an issue as to whether or not versioning should be used in the IDN dictionaries in support of the switching process. Ideally,

the switching process should be as simple and straight-forward as possible. This requirement, and the run-time nature of the IDN dictionaries tend to support the simplest of approaches:

- Create a new dictionary at each node.
- Activate the new dictionary as of a given date and time.
- Switch query processing to the new dictionary based on the query's date-time stamp.

Thus dictionary switching would be handled exactly as database switching is handled.

The decision on whether or not to switch to different dictionaries, however, must take into account how the IDN query processing system binds to the various dictionaries in the IDN. The D2Q and D1* Nodes have three common characteristics which effect the binding strategy:

- The data-oriented metadata resident in those dictionaries is static.
- The processing at these nodes involve a great deal of dictionary lookup processing.
- There is a great deal of data manipulation to be supported at these nodes.

These characteristics suggest that the data-oriented metadata in the dictionaries at these nodes should be "compiled" and placed in primary memory. If this approach is taken, then there must be a "compilation" utility, and the switching utility would actually switch query processing from using one memory area to another, rather than the dictionaries per se. If this is the case, then the

"compiled meta" could be distributed instead of the dictionary descriptors in IRDS import/export formatted files.

The situation is different for D1 Nodes. The D1 nodes do not have the dictionary lookup and data manipulation requirements of the D1* and D2Q nodes. Furthermore, the D1 dictionaries are more volatile, since query development may be an ongoing activity. This has several important consequences:

- It is not necessary to "compile" data-oriented metadata and store it in memory. (This metadata would consist of the LUNS for each user; the basic data-type information for each CDR-ELEMENT in the D1 dictionary, and the LUNS to CDR-ELEMENT correspondence.)

- It is not desirable to "compile" the entire D1 dictionary into memory. Stored queries will be space consuming, and they are relatively volatile but their usage level is low relative to the metadata at the D2Q and D1 nodes.

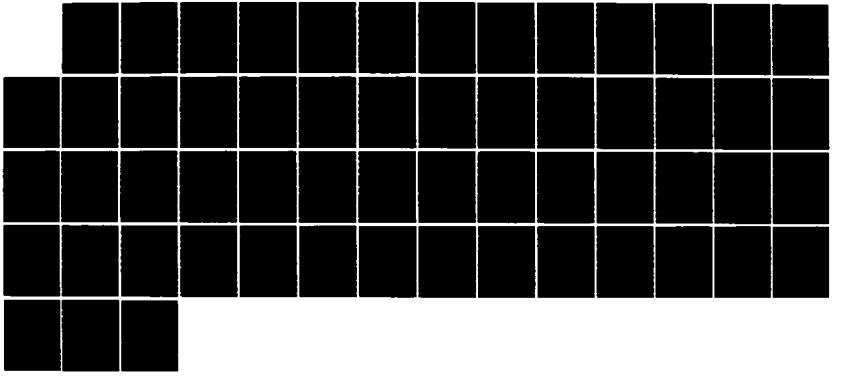
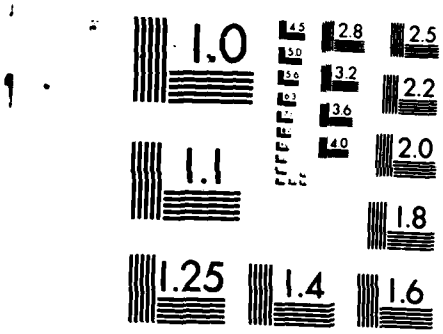
In this case, the simplicity of the "old dictionary / new dictionary switch" seems like the appropriate approach. However, it is possible that the old dictionaries may continue to be maintained while the new dictionaries are pending activation. Unless special measures are taken, it is conceivable that the old dictionary may contain some queries not present in the new dictionary. Since the IDN is supposed to support 24 hour operation and is worldwide in scope, it is unlikely that query maintenance could be shut down and cutover on D1 dictionaries accomplished when no maintenance is being performed.

Given a small enough time window, however, it may be acceptable to shut off the saving of user queries. This still allows users to use existing queries, and develop ad hoc queries (possibly based on saved queries.) Thus the query user may be inconvenienced, but not prevented from doing his work. If the User Node has available

temporary storage, it might be possible to avoid even this level of inconvenience, by holding the "saved" query at the User Node, until such time as the supporting D1 Nodes are again available to accept query updates.

AD-A173 344 DESIGN CONCEPTS FOR DATA BASE UTILITIES(U) AOG SYSTEMS CORP HARVARD MA APR 86 RADC-TR-86-48 F30602-84-C-0011 5/5

UNCLASSIFIED F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(This page left intentionally blank)

APPENDIX I

DETAILED DESCRIPTIONS OF MAJOR INTER-SUBSYSTEM DATA FLOWS

This appendix provides detailed data descriptions for the major inter-subsystem data flows in the IDN. These data descriptions are abstract in nature, i.e., they only define data structures and elements. Detailed lengths, formats, and values are not shown.

The following conventions are used:

::= means the data item on the left decomposes into the data items on the right. Thus

```
x ::= a b c d
```

means that x is composed of the sequence of items a, b, c and d.

{x} means that x repeats 0 to n times

[x] means that x is optional

<x> indicates that x is an element

x|y means that either x or y will occur. Thus:

```
X ::= x1 | x2 | ... | xn
```

means that X is a case structure with n alternatives x1, x2, ... xn.

Comments are blocked off by "/*" and "*/"

Where an element may take on a specific value of an enumerated set of values, the logical values are identified in comments.

This appendix is divided into sections. Within any section, data items are listed according to the top-down decomposition. Where an item is decomposed in another section, the section number appears on the far right. The top levels of definition are purely for documentation purposes.

Note that the correspondence between the data flow definitions in this appendix and those found in the Subsystems Specifications is approximate. The reason for this is that many data flows as defined in the Subsystems Specifications conform to a common format. In these cases, the possible values for specific elements are used to indicate the correspondence to the Subsystems Specifications.

0. Index

/* This level of description is used to identify the organization of data descriptions in this document. It also illustrates the format used throughout this document. */

Data Item	Xref
Data-description ::=	
IDN-intersystem-data-flow	
| Shared-data-description	
IDN-intersystem-data-flow ::=	
inter-processor-data-flow	
| intra-processor-data-flow	
intra-processor-data-flow ::= /* not expanded in this document */	
inter-processor-data-flow ::=	
User-Node-to-D1	1
| D1-to-D1	2
| D1-to-D2Q	3
| D2Q-to-D2Q	4
| D2Q-to-D1*	5
| D1*-to-Data-Node	6
| Data-Node-to-D1*	7
| D1*-to-D2Q	8
| D1*-to-D1*	9
| D2Q-to-D1	10
| D1*-to-D1	11
| D1-to-User-Node	12
Shared-data-description ::=	
tokenized-query	13
| intermediate-query	14
| common-query-syntax-primitives	15
| query-control-block	16
| common-data-items	17

1. User-Node to Dl Data Flow Definitions

Data Item	Xref
User-Node-to-Dl ::=	
query-maintenance	
| Dl-IRDS-query	
| IDN-query-transactions	
query-maintenance ::=	
Save-query-transaction	
| Delete-query-transaction	
| Retrieve-query-transaction	
| Check-query-transaction	
Dl-IRDS-query ::=	
LIST-transaction	
| SHOW-transaction	
IDN-query-transactions ::=	
run-IDN-query-transaction	
| kill-IDN-query-transaction	
| IDN-query-status-transaction	
| send-IDN-query-output-transaction	
| redirect-response-transaction	
Save-query-transaction ::=	
IDN-message-header	17
query-id	17
<query-state> /* tokenized | validated */	
tokenized-query	13
[intermediate-query] /* if query-state = validated */	14
Delete-query-transaction ::=	
IDN-message-header	17
query-id	17
Retrieve-query-transaction ::=	
IDN-message-header	17
query-id	17
Check-query-transaction ::=	
IDN-message-header	17
query-id	17
tokenized-query	13
LIST-transaction ::=	
IDN-message-header	17
<entity-type-name> /* QUERY | ELEMENT | FUNCTION */	
<response-destination>	
{ scan-mask }	
{ restriction-criteria }	

```
SHOW-transaction ::=
     IDN-message-header                                          17
     <entity-type-name>  /* QUERY | ELEMENT | FUNCTION */
     <response-destination>
     { scan-mask }
     { restriction-criteria }
     { sequence-argument }
     { display-option }

restriction-criteria ::=
          attribute-restriction
     |    relationship-existence-restriction
     |    related-to-restriction
     |    not-operator                                          15
          restriction-criteria
     |    restriction-criteria
          binary-boolean-operator                               15
          restriction-criteria

attribute-restriction ::=
          <attribute-type-name>
          relational-operator                                   15
          <attribute-value>

relationship-existence-restriction ::=
          < relationships-exists-predicate>
          < related-to-entity-type >

related-to-restriction-criteria ::=
          <related-to-predicate>
          <related-to-entity-name>

sequence-argument ::=
          < argument-type > /* attribute-type | name */
          < argument >      /* attribute-type-name | null */
          < asc-or-desc >   /* ASCENDING | DESCENDING */

display-option ::=
          <ALL-operator>
     |    <attributes-only-operator>
     |    display-relationships-option
     |    <attribute-type-name>

display-relationships-option ::=
          <relationships-indicator>
          <relationship-qualifier>
          /* ALL | relationship-type-name */

run-IDN-query-transaction ::=
          run-tokenized-query-transaction
     |    run-intermediate-query-transaction
```

```
kill-IDN-query-transaction ::=
     IDN-message-header                                    17
     query-id                                              17

IDN-query-status-transaction ::=
     IDN-message-header                                    17
     query-id                                              17

send-IDN-query-output-transaction ::=
     IDN-message-header                                    17
     query-id                                              17
     [ <response-destination> ]

redirect-response-transaction ::=
     IDN-message-header                                    17
     query-id                                              17
     <response-destination>

run-tokenized-query-transaction ::=
     IDN-message-header                                    17
     query-id                                              17
     tokenized-query                                       13

run-intermediate-query-transaction ::=
     IDN-message-header                                    17
     query-id                                              17
     /* assumes intermediate query is saved at D1 */
```

2. Dl to Dl Data Flow Definitions

Data Item	Xref

```
Dl-to-Dl ::=
          Save-query-transaction                                    1
     |    Delete-query-transaction                                  1
     |    LUNS-maintenance

LUNS-maintenance ::=
/*   Any maintenance to LOCAL-ELEMENT-NAMEs and their relationships
     to CDR-ELEMENTS must be synchronized across Dl nodes which support
     a given User Node. */
          IDN-message-header                                        17
          IRDS-dict-maint-command
          { IRDS-dict-maint-command }

IRDS-dict-maint-command ::=
          add-entity-command
     |    modify-entity-command
     |    delete-entity-command
     |    add-relationship-command
     |    modify-relationship-command
     |    delete-relationship-command
     |    modify-access-name-command
     |    modify-descriptive-name-command

add-entity-command ::=  /* reference IRDS dpANS */

modify-entity-command ::=  /* reference IRDS dpANS */

delete-entity-command ::=  /* reference IRDS dpANS */

add-relationship-command ::=  /* reference IRDS dpANS */

modify-relationship-command ::=  /* reference IRDS dpANS */

delete-relationship-command ::=  /* reference IRDS dpANS */

modify-access-name-command ::=  /* reference IRDS dpANS */

modify-descriptive-name-command ::=  /* reference IRDS dpANS */
```

3. D1 to D2Q Data Flow Definitions

Data Item	Xref
D1-to-D2Q ::=	
validate-intermediate-query	
| kill-IDN-query-transaction	1
| IDN-query-status-transaction	1
| redirect-response-transaction	1
| purge-QCB-SQCB-QEG-transaction	
| final-output-held-at-node-msg	
| final-output-purged-from-node-msg	
validate-intermediate-query ::=	
IDN-message-header	17
query-id	17
intermediate-query	14
purge-QCB-SQCB-QEG-transaction ::=	
IDN-message-header	17
query-id	17
final-output-held-at-node-msg ::=	
IDN-message-header	17
query-id	17
<node-id>	
final-output-purged-from-node-msg ::=	
IDN-message-header	17
query-id	17
<node-id>	

4. D2Q to D2Q Data Flow Definitions

Data Item	Xref
D2Q-to-D2Q ::=	
copy-QCB	
| query-completion-message	
| processor-role-change-message	
/* following are forwarded from D2Q/C to each D2Q/S */	
| subquery-status-change-message	8
| kill-IDN-query-transaction	1
| redirect-response-transaction	1
| restart-query-directive	
| hold-query-directive	
copy-QCB ::=	
/* Controlling D2Q sends copy of QCB to shadow D2Qs prior to query initiation */	
IDN-message-header	17
QCB	16
query-completion-message ::=	
IDN-message-header	17
query-id	1
processor-role-change-message ::=	
IDN-message-header	17
query-id	17
<IDN-node-id>	
<upgrade-or-downgrade>	
restart-query-directive ::=	
IDN-message-header	17
query-id	1
hold-query-directive ::=	
IDN-message-header	17
query-id	1

5. D2Q to Dl* Data Flow Definitions

Data Item	Xref
D2Q-to-Dl* ::=	
canonical-subquery	
| subquery-directive	
canonical-subquery ::=	
/* Sent from D2Q to Dl* prior to initiation */	
IDN-message-id	17
query-header	17
SQCB	16
SQIB	16
input-file-description	16
{ input-file-description }	16
output-file-description	16
{ output-file-description }	16
subquery-directive ::=	
IDN-message-id	17
/* where message-id identifies one of the following :	
start-subquery-transaction	
| stop-subquery-transaction	
| kill-subquery-transaction	
| verify-receipt-of-SQCB-msg	
| send-output-directive	
| subresponse-status-request	
| transmit-response-to-user	
| purge-SQCB */	
query-id	17
<subquery-id>	17

6. D1* to Data Node Data Flow Definitions

```
Data Item                                                    Xref

D1*-to-D2Q ::=
          NDRL-SQCB
     |    data-node-processing-directive

NDRL-SQCB ::=
          IDN-message-header                                 12
          query-header                                       12
          <subquery-id>
          NDRL-subquery

NDRL-subquery ::=
/* character string input, which is broken down as follows */
          NDRL-select-clause
          NDRL-from-clause
          NDRL-where-clause
          NDRL-output-clause

NDRL-select-clause ::= /* ref IDN subsystem 9 */

NDRL-from-clause ::= /* ref IDN subsystem 9 */

NDRL-where-clause ::=  /* ref IDN subsystem 9 */

NDRL-output-clause ::= /* ref IDN subsystem 9 */

data-node-processing-directive ::=
          IDN-message-header                                 12
          query-header                                       12
          <subquery-id>
          /* directives: abort, halt, resume */
```

7. Data Node to Dl* Data Flow Definitions

Data Item	Xref
Data-Node-to-Dl* ::=	
Data-Node-subquery-response	
| Data-Node-subquery-diagnostics	
Data-Node-subquery-response ::=	
IDN-message-header	17
query-id	17
<subquery-id>	
output-file-description	16
response-data	17
Data-Node-subquery-diagnostics ::=	
IDN-message-header	17
query-id	17
<subquery-id>	
<diagnostic-message-id>	
/* includes execution error,	
security violation,	
subresponse over quota	
response composition error	
response composition complete.	
subresponse status data */	
<severity-level>	
<variable-data>	
/* includes subresponse size for over quota message,	
specifics for execution error and security violation */	

8. D1* to D2Q Data Flow Definitions

```
Data Item                                                    Xref

D1*-to-D2Q ::=
          subquery-status-message
     |    subresponse                                        9
          /* exception: subresponses normally transmitted
             between D1* Nodes */
     |    input-queue-full-message

subquery-status-message ::=
          IDN-message-header                                 17
          query-id                                           17
          <subquery-id>
          <subquery-status>
          /* ***********************************************
          subquery-status =
               subresponse-received
          |    subquery-accepted
          |    subquery-initiated
          |    subquery-completed
          |    subquery-terminated
          |    subquery-backout
          |    subquery-stopped
          |    subquery-restarted
          ********************************************** */
          subquery-status-data

input-queue-full-message ::=
          IDN-message-header

subquery-status-data ::=
          subresponse-received-data
     |    subquery-accepted-data
     |    subquery-initiated-data
     |    subquery-completed-data
     |    subquery-terminated-data
     |    subquery-backout-data
     |    subquery-stopped-data
     |    subquery-restarted-data

subresponse-received-data ::=
          <subresponse-received-date-time>
          <file-description-id>        /* ref QCB */
          <local-physical-file-id >    /* ref QCB */
          <file-size-in-tuples>        /* ref QCB */
          <file-size-in-chars>         /* ref QCB */
          /* file-sizes used to cross-check all data received */

subquery-accepted-data ::=
          <date-time-accepted>
```

```
subquery-initiated-data ::=
          <date-time-initiated>

subquery-completed-data ::=
          <date-time-completed>
          output-file-description                              16

subquery-terminated-data ::=
          <date-time-terminated>
          <cause-of-termination>
          subquery-diagnostics

subquery-backout-data ::=
          <backout-date-time>
          subquery-diagnostics

subquery-stopped-data ::= /* i.e., put on hold, pending restart */
          <date-time-stopped>
          subquery-diagnostics

subquery-restarted-data ::=
          <date-time-restarted>
          subquery-diagnostics

subquery-diagnostics ::=
          subquery-diagnostic-message
          { subquery-diagnostic-message }

subquery-diagnostic-message ::=
          <diagnostic-message-id>
          /* unable to locate database input
             unable to translate NANS to DANS
             unable to associate DANS to database record-type
             invalid data element restriction
             invalid SQCB input
             subquery execution error
             security violation on subquery
             subresponse exceeds quota
             target database error
             user abort initiated
             subquery translation error
             (also ref section 7.)
          */
          <severity-level>
          <variable-data>
```

9. Dl* to Dl* Data Flow Definitions

Data Item	Xref
Dl*-to-Dl* ::=	
subresponse	
| semijoin-response	
subresponse ::=	
IDN-message-header	17
query-id	17
<subquery-id>	
subquery-sequence	
<file-description-id>	
/* a.k.a subresponse-id in Subsystems Specifications */	
<file-size-in-tuples>	
<file-size-in-chars>	
response-data	17
subquery-sequence ::=	
<subquery-number-in-sequence>	
<total-subqueries-in-sequence>	
/* i.e., n out of m subqueries */	
semijoin-response ::=	
IDN-message-header	17
query-id	17
<subquery-id>	
<file-size-in-tuples>	
<file-size-in-chars>	
response-relation-description	
join-with-relation-description	
response-data	17
response-relation-description ::=	
relation-description	16
join-with-relation-description ::=	
relation-description	16

10. D2Q to D1 Data Flow Definitions

Data Item	Xref
D2Q-to-D1 ::=	
query-status-message	
| kill-query-confirmation-message	
| redirect-response-confirmation-message	
| send-response-confirmation-message	
| composite-response	11
/* composite response normally sent from D1* */	
query-status-message ::=	
IDN-message-header	17
query-id	17
<query-status>	
status-data	
status-data ::=	
<date-time-submitted>	
[<date-time-ended>	
reason-for-end]	
[<response-size>]	
[<percent-complete-estimate>]	
kill-query-confirmation-message	
IDN-message-header	17
query-id	17
redirect-response-confirmation-message	
IDN-message-header	17
query-id	17
send-response-confirmation-message	
IDN-message-header	17
query-id	17

11. Dl*-to-Dl Data Flow Definitions

Data Item	Xref
Dl*-to-Dl ::=	
composite-response	
composite-response ::=	
IDN-message-header	17
query-id	17
response-header	
cell-response	
{ cell-response }	
response-header ::=	
<response-size-in-tuples>	
<response-size-in-chars>	
cell-mapping	
{ cell-mapping }	
cell-mapping ::=	
<cell-id>	
<horizontal-offset>	
<vertical-offset>	
<horizontal-dimension>	
<vertical-dimension>	
{ line-mapping }	
{ literal-mapping }	
cell-response ::=	
cell-response-header	
response-data	17
cell-response-header ::=	
<cell-sequence>	
<cell-id>	
<number-of-tuples>	
line-mapping ::=	
<orientation>	
<horizontal-offset>	
<vertical-offset>	
<skip-factor>	
<repetition-factor>	
{ data-mapping }	
literal-mapping ::=	
<literal>	
{ <mapping-attribute> }	
/* e.g., bold, underline, reverse-video, font, etc. } */	

```
data-mapping ::=
        <response-field-number>
        <horizontal-offset>
        <edit-mask>                    /* i.e., picture */
        {<mapping-attribute>}
```

12. D1 to User-Node Data Flow Definitions

Data Item	Xref
D1-to-User-Node ::=	
composite-response	11
| LIST-response	
| SHOW-response	
LIST-response ::=	
IDN-message-header	
response-header	11
cell-response	11
SHOW-response ::=	
IDN-message-header	
response-header	11
cell-response	11

```
/* LIST and SHOW transactions will use same data exchange
   formats as IDN queries.  Thus mapping to user terminal
   device will be consistent.
*/
```

13. Tokenized Query

Data Item **Xref**

```
/* tokenized-query                                                    */
/*                                                                    */
/* This is the form of the query which is produced at the user        */
/* node.  The user node performs lexical analysis and  basic          */
/* syntax validation.  Assuming the query passes this level           */
/* of validation, the query is passed on to the associated D1         */
/* processor in this form.                                            */
/*                                                                    */
/* The form presented here is an abstract syntax.  Specifically,      */
/* we defer issues related to exact format.  For example, we do       */
/* not specifically require encoding into reverse polish notation,    */
/* nor do we identify exact delimiters or separators.                 */
/*                                                                    */
/* The query is defined a logical form.  It consists of one or        */
/* more "cells", or 2 dimensional areas.  Cells are either disjoint   */
/* or nested (i.e., a cell may contain other cells).  Cells may       */
/* one or more lines.  Each line is a list of one or more form-       */
/* elements.  A line is a logical construct, and may have either      */
/* a horizontal or vertical orientation.                              */
/*                                                                    */

tokenized-query ::=
          tokenized-cell
          { tokenized-cell }

tokenized-cell ::=
          begin-cell-marker                                    15
          < orientation >     /* horizontal | vertical */
          <line-repetition-factor>
          cell-mapping-attributes  /* implementation
                                      dependent */
          tokenized-cell-contents
          end-cell-marker                                      15

tokenized-cell-contents ::=
          tokenized-line-or-cell
          { tokenized-line-or-cell }

tokenized-line-or-cell ::=
          tokenized-line
     |    tokenized-cell

tokenized-line ::=
          <line-delimiter>
          <line-id>
          line-mapping-attributes /* implementation
                                     dependent */
          { tokenized-form-element }
```

```
tokenized-form-element ::=
          <label>
          <form-element-length>
          <form-element-data-type>   /* Default = string */
          <form-element-mapping-attributes>
                        /* implementation dependent */
          [ <form-element-picture> ]
          [   tokenized-selection-spec
            | tokenized-computational-spec ]
          /* if no tokenized-selection-spec or
             tokenized-computational-spec follows,
             it is implicitly a selection spec
             requesting ALL */

tokenized-selection-spec ::=
          <select-delimiter>
          /* Like "WHERE" in SQL syntax, possibly encoded.
             identifies beginning of restriction */
          tokenized-expression

tokenized-computational-spec ::=
          <assignment-delimiter>
          tokenized-expression

tokenized-expression ::=
          <label> /* alpha followed by alphanumeric or sep char */
        | <numeric-constant> /* normal integer, decimal, float*/
        | <string-literal>   /* in quotes */
        | tokenized-function-expression
        | tokenized-arithmetic-expression
        | tokenized-relational-expression
        | tokenized-boolean-expression
        | tokenized-existential-expression
        | tokenized-nested-expression
        | tokenized-virtual-key-expression
        | tokenized-set-expression

tokenized-function-expression ::=
          <label>
          <left-parentheses>
          { <label> }  /* identifying other form elements */
          <right-parentheses>

tokenized-arithmetic-expression ::=
          tokenized-unary-arithmetic-expression
      |   tokenized-binary-arithmetic-expression

tokenized-relational-expression ::=
          tokenized-expression
          relational-operator                                  15
          tokenized-expression
```

```
tokenized-boolean-expression ::=
          tokenized-unary-boolean-expression
    |     tokenized-binary-boolean-expression

tokenized-existential-expression ::=
          <label>
          [ <not-operator> ]
          <exists-indicator>

tokenized-nested-expression ::
          <left-parentheses>
                    tokenized-expression
                    right-parentheses

tokenized-unary-arithmetic-expression ::=
                    simple-tokenized-unary-arithmetic-expression
                  | compound-tokenized-unary-arithmetic-expression

simple-tokenized-unary-arithmetic-expression ::=
                    unary-arithmetic-operator
                    tokenized-label-or-function

tokenized-label-or-function ::=
                    label
                  | tokenized-function-expression

compound-tokenized-unary-arithmetic-expression ::=
                    unary-arithmetic-operator
                    left-parentheses
                    tokenized-arithmetic-expression
                    right-parentheses

tokenized-binary-arithmetic-expression ::=
                    tokenized-arithmetic-expression
                    binary-arithmetic-operator
                    tokenized-arithmetic-expression

tokenized-unary-boolean-expression ::=
                    not-operator
                    tokenized-expression

tokenized-binary-boolean-expression ::=
                    tokenized-expression
                    binary-boolean-operator
                    tokenized-expression

tokenized-set-expression ::=
                    label
                    [ not-operator ]
                    in-set-operator
                    begin-set-delimiter
                    { literal }
                    end-set-delimiter
```

```
tokenized-virtual-key-expression ::=
                    virtual-key-set-expression
                  | virtual-key-equality
                  | virtual-key-inequality

virtual-key-set-expression ::=
                    virtual-key-literal
                    [ not-operator ]
                    in-set-operator
                    begin-set-delimiter
                    { literal }
                    end-set-delimiter

virtual-key-equality ::=
                    virtual-key-literal
                    equals-operator
                    literal

virtual-key-inequality ::=
                    virtual-key-literal
                    not-equal-operator
                    literal
```

14. Intermediate Query

Data Item **Xref**

```
/* intermediate-query                                                  */
/*                                                                     */
/* This is the form of the query which is produced by the Receive      */
/* Query Subsystem (Subsystem 2) and which is processed by Query       */
/* Semantic Validation (Subsystem 3).  In this form of the query,      */
/* all labels are replaced by references to the query symbol table.    */
/* The meaning of all labels is determined insofar as whether          */
/* a label is local to a query, or whether it refers to a              */
/* catalogued FUNCTION, or CDR-ELEMENT.                                */
/*                                                                     */
/* The intermediate-query is part of the Query Control Block (QCB).    */

intermediate-query ::=
          symbol-table
          normalized-cell
          { normalized-cell }

symbol-table ::=
          symbol-table-header
          symbol-table-body
          symbol-usage-table

symbol-table-header ::=
          number-of-symbols
          symbol-table-size

symbol-table-body ::=
          { symbol-table-entry }

symbol-table-entry ::=
          <label>
          <label-type>
          <label-usage-offset>
          label-defn

/* ********************************************************************

label-type =  local-user-name | function-name | query-local-label
where
     local-user-name is assigned-access-name of a LOCAL-ELEMENT-NAME
     function-name is assigned-access-name of a FUNCTION
     query-local-label means neither of the above

******************************************************************** */

label-defn ::=
          query-label-defn  /* if label-type = query-local-label */
     |    local-user-name-defn  /* if label-type = local-user-label */
```

```
symbol-usage-table ::=
          { symbol-usage }

symbol-usage ::=
          <usage-count>
          usage-ptr
          { usage-ptr }

usage-ptr ::=
          <cell-id>
          <line-id>
          <form-element-within-line>
          <token-within-form-element>

query-label-defn ::=
          <format>
          <length>
          <picture>

local-user-name-defn ::=
          <format>
          <length>
          <picture>
          <corresponding-network-access-name>

normalized-cell ::=
          begin-cell-marker                                    15
          <orientation>
          <line-repetition-factor>
          cell-mapping-attributes
               /* implementation dependent */
          normalized-cell-contents
          end-cell-marker                                      15

normalized-cell-contents ::=
          normalized-line-or-cell
          { normalized-line-or-cell }

normalized-line-or-cell ::=
          normalized-line
     |    normalized-cell

normalized-line ::=
          <line-delimiter>
          <line-id>
          line-mapping-attributes
               /* implementation dependent */
          normalized-form-element
          { normalized-form-element }
```

```
normalized-form-element ::
          <symbol-table-id>
          <form-element-length>
          <form-element-data-type>     /* Default = string */
          form-element-mapping-attributes
                   /* implementation dependent */
          [ <form-element-picture> ]
          [   normalized-selection-spec
            | normalized-computational-spec ]
            /* if no normalized-selection-spec or
               normalized-computational-spec follows,
               it is implicitly a selection spec requesting ALL */

normalized-selection-spec ::=
          <select-delimiter>
          /* Like "WHERE" in SQL syntax, possibly encoded.
             identifies beginning of restriction */
          normalized-expression

normalized-computational-spec ::=
          <assignment-delimiter>
          normalized-expression

normalized-expression ::=
          form-element-reference
        | <numeric-constant>  /* normal integer, decimal, float*/
        | <string-literal>    /* in quotes */
        | normalized-function-expression
        | normalized-arithmetic-expression
        | normalized-relational-expression
        | normalized-boolean-expression
        | normalized-existential-expression
        | normalized-nested-expression
        | normalized-virtual-key-expression
        | normalized-set-expression

form-element-reference ::=
          <form-element-marker>
          <symbol-table-id>

normalized-function-expression ::=
          <function-marker>
          <symbol-table-id>
          { <form-element-reference> }
          /* for function arguments */

normalized-arithmetic-expression ::=
          normalized-unary-arithmetic-expression
        | normalized-binary-arithmetic-expression

normalized-relational-expression ::=
          normalized-expression
          relational-operator
          normalized-expression
```

```
normalized-boolean-expression ::=
          normalized-unary-boolean-expression
     |    normalized-binary-boolean-expression

normalized-existential-expression ::=
          <form-element-reference>
          [ <not-operator> ]
          <exists-indicator>

normalized-nested-expression ::=
          <left-parentheses>
          normalized-expression
          <right-parentheses>

normalized-unary-arithmetic-expression ::=
          simple-normalized-unary-arithmetic-expression
     |    compound-normalized-unary-arithmetic-expression

simple-normalized-unary-arithmetic-expression ::=
          unary-arithmetic-operator
          [ <form-element-reference >
          | normalized-function-expression ]

compound-normalized-unary-arithmetic-expression ::=
          <unary-arithmetic-operator>
          <left-parentheses>
          normalized-arithmetic-expression
          <right-parentheses>

normalized-binary-arithmetic-expression ::=
          normalized-arithmetic-expression
          <binary-arithmetic-operator>
          normalized-arithmetic-expression

normalized-unary-boolean-expression ::=
          <not-operator>
          normalized-expression

normalized-binary-boolean-expression ::=
          normalized-expression
          <binary-boolean-operator>
          normalized-expression

normalized-set-expression ::=
          <form-element-reference>
          [ <not-operator> ]
          <in-set-operator>
          <begin-set-delimiter>
          { <literal> }
          <end-set-delimiter>

normalized-virtual-key-expression ::=
          virtual-key-set-expression
     |    virtual-key-equality
     |    virtual-key-inequality
```

```
virtual-key-set-expression ::=
          <virtual-key-operator>
          [ <not-operator> ]
          <in-set-operator>
          <begin-set-delimiter>
          { <literal> }
          <end-set-delimiter>

virtual-key-equality ::=
          <virtual-key-operator>
          <equals-operator>
          <literal>

virtual-key-inequality ::=
          <virtual-key-operator>
          <not-equal-operator>
          <literal>
```

15. Query Syntax - Common Data Items

Data Item **Xref**

```
begin-cell-marker ::=
                    begin-cell-delimiter
                    cell-id

end-cell-marker ::=
                    end-cell-delimiter
                    cell-id

unary-arithmetic-operator ::=
                    plus-operator
                  | minus-operator

binary-arithmetic-operator ::=
                    plus-operator
                  | minus-operator
                  | times-operator
                  | divided-by-operator

binary-boolean-operator ::=
                    and-operator
                  | or-operator

relational-operator ::=
                    equals-operator
                  | not-equal-operator
                  | greater-than-operator
                  | less-than-operator
                  | greater-than-or-equal-operator
                  | less-than-or-equal-operator
                  | not-greater-than-operator
                  | not-less-than-operator

literal ::=
                    label
                  | numeric-literal
                  | string-literal
```

16. Query Control Block

Data Item **Xref**

```
/* query control block                                                    */
/*                                                                        */
/* The query control block is an aggregate of many different              */
/* structures which control the execution of the query.  Its              */
/* major components are:                                                  */
/*                                                                        */
/*   The QCB-header - which contains global control information           */
/*                                                                        */
/*   The message queue - which contains all administrative and            */
/*        diagnostic messages                                             */
/*                                                                        */
/*   The intermediate query - the form of the query after                 */
/*        completing the validation at the D1 node.  If need be           */
/*        the entire query control block can be regenerated by            */
/*        revalidating the intermediate query                             */
/*                                                                        */
/*   The execution graph - which is the set of all subquery               */
/*        control blocks                                                  */
/*                                                                        */
/*   Subquery control blocks (SQCBs)- another aggregate which             */
/*        consists of                                                     */
/*                                                                        */
/*        A header (to monitor & control the subquery).                   */
/*                                                                        */
/*        File descriptions (primarily for temporary files)               */
/*                                                                        */
/*        Relation descriptions (defining how data is to be               */
/*             accessed or transmitted)                                   */
/*                                                                        */
/*        Subquery Instruction Blocks (SQIBs)-(the instructions used      */
/*             to construct a physical database query.)                   */
/*                                                                        */
/*        Relation descriptions and subquery bodies may be                */
/*        shared by different subqueries, because the same                */
/*        data types may reside at different nodes.                       */
/*                                                                        */

query-control-block ::=
          query-control-block-header
          messages-queue
          intermediate-query                                        14
          query-execution-graph

query-control-block-header ::=
          query-header                                              17
          monitors
```

```
monitors ::=
          <number-of-D2Q-nodes>
          node-assignment
          { node-assignment }

node-assignment ::=
          <IDN-node-id>
          <node-responsibility>          /* controlling | alternate */

     /*   ********************************************************
          Normally 1st IDN-node in the monitors array is the
          controlling D2Q.  However, should this have to change,
          attempts to reassign the controlling node responsibility
          for the particular query are made in the order of this
          list. In that case, the values of node-responsibility
          are changed accordingly.
          ********************************************************   */

query-execution-graph ::=
          execution-graph-header
          SQCBs
          SQIBs
          file-descriptions
          relation-descriptions

execution-graph-header ::=
          <query-initiation-date-time>
          <query-completion-date-time>
          <number-of-pending-subqueries>
          <number-of-holding-subqueries>
          <number-of-executing-subqueries>
          <number-of-completed-subqueries>
          <number-of-unique-D1*s>

SQCBs ::=
          <number-of-SQCBs>
          SQCB
          { SQCB }

SQIBs ::=
          <number-of-SQIBs>
          SQIB
          { SQIB }

file-descriptions ::=
          <number-of-file-descriptions>
          file-description
          { file-description }

relation-descriptions ::=
          <number-of-relation-descriptions>
          relation-description
          { relation-description }
```

```
SQCB ::=
          subquery-control-block-header
          <SQIB-id>
          <number-of-input-files>
          {<input-file-description-id>}
          <number-of-output-files>
          {<output-file-description-id>}

subquery-control-block-header ::=
          <subquery-id>
          <initiation-date-time>
          <termination-date-time>
          <assigned-IDN-node>      /* i.e., D1* */
          <subquery-state>
          /* ********************************************************
          subquery-state =
          pending | waiting | executing | hold | completed | killed

          where
               pending ==> not yet started
               waiting ==> waiting on input files
               executing ==> running at data node
               hold ==> output ready
               completed ==> normal completion
               killed ==> stopped by user or system
               abend ==> terminated abnormally
          ******************************************************** */
          data-source-control-block

data-source-control-block ::=
          <number-of-cdrs>
          data-source-descr
          { <data-source-descr> }

data-source-descr ::=
          <cdr-network-access-name>
          <assigned-idn-node-name>

SQIB ::=
          /* For database only subqueries,
             database/subresponse subqueries,
             join/semijoin directives,
             and response composition program
          */
          SQIB-id
          { relation-description-id }
          /* points to relation description */
          subquery-symbol-table
          subquery-operations

subquery-symbol-table ::= /* similar to intermediate query */

subquery-operations ::=
          subquery-operation
          { subquery-operation }
```

```
subquery-operation
          join-operation
     |    select-operation     /* PROJECT operation embedded in SELECT */
     |    sort-operation       /* for response composition program */
     |    merge-operation      /* for response composition program */

join-operation ::=
          <join-operator>      /* natural join, semi-join append etc. */
          <file-id-1>          /* ref SQCB symbol table */
          <file-id-2>          /* ref SQCB symbol table */
          <attribute-id>
          { <attribute-id> }

select-operation ::=  /* as in SQL */

sort-operation ::=
          <sort-operator>
          <file-id>            /* ref SQCB symbol table */
          sequence-arg

sequence-arg ::=
          <attribute-id>
          <ascend-or-descend>

file-description ::=
          <file-description-id>
          <SQCB-id>
          <input-or-output>
          case-of-input-or-output

case-of-input-or-output ::=
          input-file-description
     |    output-file-description

input-file-description ::=
          <input-file-type>              /* DB | TEMP */
          case-of-input-file-type

case-of-input-file-type ::=
          db-file-description            /* a "logical db file" */
     |    temp-file-description          /* from another subquery */

db-file-description ::=
          <relation-description-id>
```

```
temp-file-description ::=
          <local-physical-file-id>   /* operating system name or id */
          <output-file-description-id>
          <relation-description-id>

output-file-description ::=
          <local-physical-file-id>   /* operating system name or id */
          <file-size-in-tuples>
          <file-size-in-chars>
          <number-of-relation-descriptions>
          {<corresponding-relation-description>}
          /* normally only one relation description
            / file description, exception is final response file */
```

17. Common Data Items

Data Item **Xref**

```
IDN-message-header ::=
          <message-type-id>
          <sending-node-id>
          <target-node-id>
          <message-date/time-stamp>

query-header ::=
          query-id
          <priority>
          <security-markings>
          <mode-indicator>          /* foreground | background */
          <response-destination>
          <notify-indicator>        /* for background execution */

query-id ::=
          <origin-node-id>
          <user-id>
          <date-submitted>          /* in network universal time */
          <time-submitted>          /* in network universal time */
          <assigned-query-name>     /* user designated name */

query-diagnostics ::=
          query-diagnostic-header
          query-diagnostic-message
          { query-diagnostic-message }

query-diagnostic-header ::=
          query-id
          query-execution-mode /* foreground | background */
          return-destination   /* for delayed output */
          notify-indicator     /* for delayed output */

query-diagnostic-message ::=
          error-code
          cell-id
          line-id
          form-element-within-line
          token-within-form-element-specification
          diagnostic-variable-data

response-data ::=
          { response-record }

response-record ::=
          { response-field }
          <end-of-record-marker>
```

```
response-field ::=
          [<character-string>]
          <separator-character>
```

(This page left intentionally blank)

APPENDIX II

IDN IRDS SCHEMA

1. INTRODUCTION

The Information Resource Dictionary System (IRDS) exists at each IDN processor. The IRDS services the IDN subsystems by providing a single mechanism to store, maintain, and retrieve metadata.

Each IDN processor has a single Information Resource Dictionary (IRD) to maintain metadata. A single schema defines the types of metadata held within each IRD.

Each IDN processor fulfills one or more roles in the processing of IDN queries. These roles have been identified as the D1, D2Q, and D1* roles. These roles are defined by different support functions in the processing of IDN queries. The types of metadata within any given IRD will be restricted to the types of metadata required for the role(s) assigned to the corresponding processor. The set of types of metadata which support a particular role has been called the "processor-role-IRDS DICTIONARY-VIEW" in the Subsystems Descriptions. Thus the "D2Q-IRDS DICTIONARY-VIEW" defines the types of metadata required by a D2Q processor.

This appendix defines the schema used by the IRDS in the IDN, and the Views of the schema which correspond to each IDN processor role. The type definitions are based upon the Entity-Relationship-Attribute (ERA) formalism used in the draft proposed American National Standard for Information Resource Dictionary Systems (dpANS IRDS). This appendix presumes that the reader is familiar with the concepts and terminology of the dpANS IRDS.

The appendix is organized as follows:

Section	Title
1	Introduction
2	Entity-types
3	Relationship-types
4	Attribute-types
5	Entity-type/Attribute-type Correspondence
6	Relationship-type/Attribute-type Correspondence
7	Attribute-group-types
8	IRDS Views

2. ENTITY-TYPES

There are 29 entity-types defined in the IDN IRDS schema. They will be identified and discussed in eight groups for convenience and understandability. These groups are as follows:

- Group 1: Security Related Entity-Types
- Group 2: Query Processing Related Entity-Types
- Group 3: IDN Global Data Definition Entity-Types
- Group 4: Network Database Definition Entity-Types
- Group 5: Relational Database Definition Entity-Types
- Group 6: Hierarchical Database Definition Entity-Types
- Group 7: Inverted-list Database Definition Entity-Types
- Group 8: Basic Data Type Entity-Types

2.1 Group 1: Security Related Entity-Types

The first group consists of the following entity-types:

DICTIONARY-USER
DICTIONARY-VIEW

Both DICTIONARY-USER and DICTIONARY-VIEW are defined in the dpANS IRDS, and are required for the IRDS security facilities. At each processor, there will be a DICTIONARY-USER entity for each role assigned to the processor, and a corresponding DICTIONARY-VIEW. In addition, a DICTIONARY-USER and corresponding DICTIONARY-VIEW will be defined for the Node Administrator. Finally, at D1 Nodes, there will be a DICTIONARY-USER entity for each IDN user supported by the node. All such DICTIONARY-USER entities share a common DICTIONARY-VIEW entity, called IDN-QUERY-USER-VIEW.

2.2 Group 2: Query Processing Related Entity-Types

The second group of entity-types support different specific requirements of IDN query processing. They are as follows:

- IDN-NODE

  This entity-type is used to define the network for each IDN node. The entities of this type act as a directory of all IDN nodes.

- STORED-QUERY

  Entities of this type are used to store and control access to user-defined queries.

- ACTIVE-QUERY

  Entities of this type act as identifiers of queries being processed or controlled at a node. They correspond to a QCB at D2Q Nodes, and SQCBs at D1* Nodes.

- FUNCTION

  Entities of this type identify functions in the query "language". The IDN comes equipped with certain built-in functions (e.g., AVERAGE, SUM, etc.), and each of these have a corresponding entity in each IRD. By making these functions explicit, and the query interface reference the dictionary for function definitions, the "language" may be extended.

- LOCAL-ELEMENT-NAME

  Entities of this type define the names which a user uses to reference data elements known to IDN. Each LOCAL-ELEMENT-NAME consists of two parts: a assigned access name which has been designated by the Node administrator, and a variation-name which is the same as the user's identifier. Thus all LOCAL-ELEMENT-NAMEs with a given user's identifier in the variation-name constitute the *Local User Name Space* (LUNS) for the particular user.

2.3 Data Definition Entity-Types

The remaining five groups of entity-types are all used to define data structures.

2.3.1 Group 3: IDN Global Data Definition Entity-Types

This group consists of the following entity-types:

CDR
CDR-RECORD-TYPE
CDR-KEY
CDR-ELEMENT
VALUE-SET

These entity-types are all prefixed by "CDR". They are used to define Coherent Database Representations (or CDRs for short). A CDR is a representation of a Database as third normal form relations. Each CDR is a set of CDR-RECORD-TYPEs. CDR-RECORD-TYPEs are used to define such relations, and CDR-ELEMENTs correspond to attributes of such relations. The key for any given CDR-RECORD-TYPE is defined by a corresponding CDR-KEY, which is a set of CDR-ELEMENTs which must also be contained in the

corresponding CDR-RECORD-TYPE. The VALUE-SET entity-type is used to define Delegated Production Policy (DPP).

2.3.2 Groups 4 - 7: Target Database Definition

The next four groups of entity-types are used to define database structures visible to the IDN for each of the four basic data models: the Network Model; the Hierarchical Model; the Relational Model, and the Inverted Model. Because of the differences in semantics among these data models, separate entity-types are used to define that part of the database which is visible to the IDN. The entity-types corresponding to each of the data models identified above are prefixed by "NDB-", "HDB-", "RDB-" and "IDB-" respectively. These entity-types groups are as follows:

Group 4: Network Database Definition Entity-Types

NDB-VIEW
NDB-SET
NDB-ELEMENT

Group 5: Relational Database Definition Entity-Types

RDB-VIEW
RDB-RECORD-TYPE
RDB-ELEMENT

Group 6: Hierarchical Database Definition Entity-Types

HDB-VIEW
HDB-RECORD-TYPE
HDB-ELEMENT

Group 7: Inverted-List Database Definition Entity-Types

IDB-VIEW
IDB-RECORD-TYPE
IDB-ELEMENT

The definition of a target database to the IDN need not be a full schema. In particular, since the data query language or report writer facility provided by each individual DBMS is to be used to extract the data, specific physical database characteristics such as file sizes and indices need not be defined to the IDN. Thus the IDN is given a "VIEW" of each database, and the database declarations used by the IDN are called "datamodel-VIEWs".

The reason for using different entity-types is pragmatic. The closer the descriptions of the target database match the semantics of the corresponding DBMS data definition language (DDL), the easier it will be to provide automated mechanisms for loading these descriptors in the IRD. Furthermore, if a given database uses a very specialized DBMS with very different semantics, the present approach reduces the impact of incorporating that database into the IDN.

2.3.3 Group 8: Basic Data Type Entity-Types

The final group of entity-types define the primitive data-types used in the IDN or any of the underlying DBMSs:

INTEGER
DECIMAL
FLOAT
CHAR
VARCHAR

The data types identified here are candidate entity-types. They correspond to the data types used by SQL. The list of primitive data types is not necessarily complete. It is possible that additional data types will be defined to defined the data structures at the target databases, or to support specialized functions or data presentation in the IDN query "language". For example, both DATE and TIME might be best defined as additional basic data-types. Should the need to integrate pictorial data within IDN queries arise, various picto-graphic data types could also be defined.

3. RELATIONSHIP-TYPES

In the IDN schema, some relationship-types are used to support very specific processes in the IDN. Other relationship-types fall into groups which support other functions. Where appropriate, the relationship-types are presented in groups. In order to aid understandability, the order of presentation approximates the order of the processes supported. Unless otherwise noted, relationship-types are unsequenced. Relationship-types are also many-to-many unless otherwise noted.

3.1 Relationship-types Supporting the IDN Query User

The following relationship-type enables a user to cross-reference queries with data elements.

STORED-QUERY-USES-LOCAL-ELEMENT-NAME

The following relationship-types are used to define functions for the IDN query interface.

FUNCTION-RESULT-IS-A-INTEGER
FUNCTION-RESULT-IS-A-DECIMAL
FUNCTION-RESULT-IS-A-FLOAT
FUNCTION-RESULT-IS-A-CHAR
FUNCTION-RESULT-IS-A-VARCHAR

FUNCTION-PARAMETER-IS-A-INTEGER
FUNCTION-PARAMETER-IS-A-DECIMAL
FUNCTION-PARAMETER-IS-A-FLOAT
FUNCTION-PARAMETER-IS-A-CHAR
FUNCTION-PARAMETER-IS-A-VARCHAR

A given FUNCTION may participate in only one "RESULT-IS-A" relationship. The "PARAMETER-IS-A" relationship-types are sequenced so that the order of parameters is defined.

The following relationship-type associates LOCAL-ELEMENT-NAMEs to CDR-ELEMENTs.

LOCAL-ELEMENT-NAME-ASSIGNED-TO-CDR-ELEMENT

A given LOCAL-ELEMENT-NAME may be associated with only one CDR-ELEMENT.

3.2 Relationship-types Defining CDRs

The following relationship-type define CDRs.

CDR-SET-OF-CDR-RECORD-TYPE
CDR-RECORD-TYPE-HAS-CDR-KEY
CDR-RECORD-TYPE-SET-OF-CDR-ELEMENT
CDR-KEY-SET-OF-CDR-ELEMENT

The following relationship-types define the data-type of a given CDR-ELEMENT as perceived by the query user. (Note that the data exchange format for all CDR-ELEMENTS is VARCHAR. Thus there is no need to capture this in the IDN IRDs.)

CDR-ELEMENT-REPRESENTED-AS-INTEGER
CDR-ELEMENT-REPRESENTED-AS-DECIMAL
CDR-ELEMENT-REPRESENTED-AS-FLOAT
CDR-ELEMENT-REPRESENTED-AS-CHAR
CDR-ELEMENT-REPRESENTED-AS-VARCHAR

Note that the above relationship-types are dependent upon the data-types of the IDN query interface. The data-types identified here are tentative, taken from SQL.

The following data types are used to define Delegated Production Policy. They are used in the query decomposition process (Subsystem 5: DECOMPOSE-QUERY-INTO-CANONICAL-SUBQUERIES).

CDR-ELEMENT-HAS-VALUE-SET
VALUE-SET-IN-CDR

Finally, the following relationship-type is used to designate Virtual-Key data elements:

CDR-ELEMENT-IDENTIFIES-CDR

3.3 Relationship-types Defining Target Database Views

The following relationship-types define NDB-VIEWs.

NDB-VIEW-HAS-RECORD-OF-NDB-ELEMENT
NDB-VIEW-CONTAINS-NDB-SET
NDB-SET-HAS-RECORD-OF-NDB-ELEMENT
NDB-ELEMENT-STRUCTURE-OF-NDB-ELEMENT
NDB-ELEMENT-ARRAY-OF-NDB-ELEMENT
NDB-ELEMENT-DEPENDS-ON-NDB-ELEMENT

Note that *NDB-ELEMENTs are recursive.* The relationship-types between two NDB-ELEMENTs enable full definition of all COBOL data-structures. This definition capability may not be necessary. If not, the relationship-types could be simplified.

The following relationship-types are used to generate PICTURE clauses. The basic data-types used here may be modified as required.

NDB-ELEMENT-REPRESENTED-AS-INTEGER
NDB-ELEMENT-REPRESENTED-AS-DECIMAL
NDB-ELEMENT-REPRESENTED-AS-FLOAT
NDB-ELEMENT-REPRESENTED-AS-CHAR
NDB-ELEMENT-REPRESENTED-AS-VARCHAR

The following relationship-types define RDB-VIEWs.

RDB-VIEW-SET-OF-RDB-RECORD-TYPE
RDB-RECORD-TYPE-SET-OF-RDB-ELEMENT

RDB-ELEMENT-REPRESENTED-AS-INTEGER
RDB-ELEMENT-REPRESENTED-AS-DECIMAL
RDB-ELEMENT-REPRESENTED-AS-FLOAT
RDB-ELEMENT-REPRESENTED-AS-CHAR
RDB-ELEMENT-REPRESENTED-AS-VARCHAR

The "REPRESENTED-AS" relationship-types may be modified if required.

The following relationship-types are used to define HDB-VIEWs.

HDB-VIEW-TREE-OF-HDB-RECORD-TYPE
HDB-RECORD-TYPE-STRUCTURE-OF-HDB-ELEMENT

HDB-ELEMENT-REPRESENTED-AS-INTEGER
HDB-ELEMENT-REPRESENTED-AS-DECIMAL
HDB-ELEMENT-REPRESENTED-AS-FLOAT
HDB-ELEMENT-REPRESENTED-AS-CHAR
HDB-ELEMENT-REPRESENTED-AS-VARCHAR

The "REPRESENTED-AS" relationship-types may be modified if required.

The following relationship-types define IDB-VIEWs.

IDB-VIEW-SET-OF-IDB-RECORD-TYPE
IDB-RECORD-TYPE-STRUCTURE-OF-IDB-ELEMENT

IDB-ELEMENT-REPRESENTED-AS-INTEGER
IDB-ELEMENT-REPRESENTED-AS-DECIMAL
IDB-ELEMENT-REPRESENTED-AS-FLOAT
IDB-ELEMENT-REPRESENTED-AS-CHAR
IDB-ELEMENT-REPRESENTED-AS-VARCHAR

The "REPRESENTED-AS" relationship-types may be modified if required.

3.4 Relationship-types Defining the Correspondence to Target Databases

The following relationship-types define the correspondence between CDRs and there components, and the database-VIEWs and their components.

CDR-DERIVED-FROM-NDB-VIEW
CDR-RECORD-TYPE-DERIVED-FROM-NDB-ELEMENT
CDR-ELEMENT-DERIVED-FROM-NDB-ELEMENT

CDR-DERIVED-FROM-RDB-VIEW
CDR-RECORD-TYPE-DERIVED-FROM-RDB-RECORD-TYPE
CDR-ELEMENT-DERIVED-FROM-RDB-ELEMENT

CDR-DERIVED-FROM-HDB-VIEW
CDR-RECORD-TYPE-DERIVED-FROM-HDB-RECORD-TYPE
CDR-ELEMENT-DERIVED-FROM-HDB-ELEMENT

CDR-DERIVED-FROM-IDB-VIEW
CDR-RECORD-TYPE-DERIVED-FROM-IDB-RECORD-TYPE
CDR-ELEMENT-DERIVED-FROM-IDB-ELEMENT

The following relationship-type is used to associate a CDR with a given Data-Node.

CDR-AT-IDN-NODE

3.5 Relationship-types Defining the Static Network View

The following relationship-types define the "Static Network View".

IDN-NODE-TO-IDN-NODE
IDN-NODE-HAS-SHADOW-IDN-NODE

The IDN-NODE-TO-IDN-NODE defines the network topology. It may not be necessary to define the entire network topology at each node.

The IDN-NODE-HAS-SHADOW-IDN-NODE is used to define which nodes may act as a shadow node for any given node. This relationship-type is sequenced. The sequence of these relationships for any given IDN-NODE is the order in which shadow nodes may be selected if the primary node fails.

4. ATTRIBUTE-TYPES

The following table lists the attribute-types identified so far for the IDN IRDS Schema. Requirements for additional attribute-types may be identified in detail design.

The column headings have the following meanings:

Attribute-type-name - the primary name of the attribute-type in the IRDS Schema

Std - An asterisk in this column indicates that the attribute-type is defined in the dpANS IRDS System Standard Schema.

Sys - An asterisk here indicates that attributes of this type are system-maintained.

Com - An asterisk here indicates that the attribute-type is common to all entity-types.

Fmt - Format:
- a S indicates "String"
- a N indicates "Numeric"
- a T indicates "Text"
- a X indicates non-printable characters

Attribute-type-name	Std	Sys	Com	Fmt
ADDED-BY	*	*	*	S
ADD-PERMISSION	*			S
ADMINISTRATOR-PERMISSION	*			S
ALLOWABLE-VALUE	*			S
ARRAY-SIZE				N
COMMAND-LANGUAGE-PERMISSION	*			S
COMMENTS	*		*	T
DATE-ADDED	*	*	*	S
DEFAULT-DICTIONARY-VIEW	*			S
DELETE-PERMISSION	*			S
DESCRIPTION	*		*	S
ENTITY-TYPE-NAME	*			S
EXCLUDE-RELATIONSHIPS	*			S
HIGH-OF-RANGE	*			S
INTERMEDIATE-QUERY-CONTENT		*		X
INTERNAL-FORMAT	*			S
JUSTIFICATION	*			S
LAST-EXECUTED		*		S
LAST-MODIFICATION-DATE	*	*	*	S
LAST-MODIFIED-BY	*	*	*	S
LAST-VALIDATED		*		S
LENGTH	*			N
LIFE-CYCLE-PHASE-NAME	*			S
LOW-OF-RANGE	*			S
MODIFY-PERMISSION	*			S
MODIFY-PHASE-PERMISSION	*			S
NUMBER-OF-MODIFICATIONS	*	*	*	N
PANEL-PERMISSION	*			S
PRECISION	*			N
QCB-ADDRESS		*		N
QUERY-PERMISSION				S
QUERY-STATE		*		S
RANGE-OR-LIST				S
READ-PERMISSION	*			S
RECORD-LEVEL				N
RELATIVE-POSITION	*			N
RENAME-PERMISSION	*			S
SCALE	*			N
SCHEMA-PERMISSION-1	*			S
SCHEMA-PERMISSION-2	*			S
SCHEMA-PERMISSION-3	*			S
SCHEMA-PERMISSION-4	*			S
SCHEMA-PERMISSION-5	*			S
SUB-TYPE		*		S
TOKENIZED-QUERY-CONTENT		*		X
USAGE	*			S

5. ENTITY-TYPE / ATTRIBUTE-TYPE CORRESPONDENCE

In this section, the correspondence between entity-types and attribute-types is defined.

At this point, most entity-types in the IDN IRDS Schema have no associated attribute-types other than those which are common to all entity-types. Therefore, rather than provide a cross-reference matrix which defines the associations, only the exceptions will be identified.

The security-related entity-types DICTIONARY-USER and DICTIONARY-VIEW are defined the almost exactly as in the dpANS IRDS System Standard Schema. The only variance is that an additional attribute-type QUERY-PERMISSION has been added to DICTIONARY-USER. The QUERY-PERMISSION attribute-type is a string with the allowed values of "Y" and "N". A "Y" indicates that the user is given permission to create, maintain, delete and run IDN queries.

The ACTIVE-QUERY entity-type has one special attribute-type: QCB-ADDRESS. This attribute-type identifies where the QCB (or SQCB) for the query is being held.

The STORED-QUERY has the following attribute-types:

- QUERY-STATE

  This attribute-type indicates whether the TOKENIZED or INTERMEDIATE query is saved.

- LAST-VALIDATED

  This identifies the date and time that the query was last checked.

- LAST-EXECUTED

  This identifies the date and time the query was last run.

- TOKENIZED-QUERY-CONTENT

  This special attribute-type is used to store the tokenized query.

- INTERMEDIATE-QUERY-CONTENT

  This special attribute-type is used to store the intermediate query.

The VALUE-SET entity-type has two associated ATTRIBUTE-TYPES. The first is RANGE-OR-LIST. This attribute-type indicates whether the value set is a RANGE of values or a simple list of values. If it is a RANGE, then an attribute-group of type ALLOWABLE-RANGE is presumed to exist. If it

is a LIST, then one or more ALLOWABLE-VALUEs are presumed to exist. Note that ALLOWABLE-VALUE is a repeating attribute-type for this entity-type.

The last entity-type with any special attribute-types is the NDB-ELEMENT. The attribute-type SUB-TYPE is used to indicate whether the NDB-ELEMENT is a STRUCTURE, ARRAY or ITEM. This identifies whether there are "STRUCTURE-OF" relationships, or if there is an "ARRAY-OF" or REPRESENTED-AS relationship for the NDB-ELEMENT.

6. RELATIONSHIP-TYPE / ATTRIBUTE-TYPE CORRESPONDENCE

In the IDN IRDS Schema, the assignment of attribute-types to relationship-types falls into groups. The grouping is used to facilitate discussion. In the following tables, an asterisk next to the attribute-type-name indicates that it is a sequencing attribute-type.

The first group of relationship-types with attribute-types consists of the "FUNCTION-PARAMETER" relationship types.

Relationship-type	Associated Attribute-type(s)
FUNCTION-PARAMETER-IS-A-INTEGER	RELATIVE-POSITION LENGTH USAGE
FUNCTION-PARAMETER-IS-A-DECIMAL	RELATIVE-POSITION LENGTH SCALE PRECISION USAGE
FUNCTION-PARAMETER-IS-A-FLOAT	RELATIVE-POSITION PRECISION
FUNCTION-PARAMETER-IS-A-CHAR	RELATIVE-POSITION LENGTH USAGE
FUNCTION-PARAMETER-IS-A-VARCHAR	RELATIVE-POSITION LENGTH USAGE

In relationships of these types, the RELATIVE-POSITION attribute-type identifies the relative position of the parameter for the particular function.

The next group consists of those relationship-types which define data-structures.

Relationship-type	Associated Attribute-type(s)
NDB-SET-HAS-RECORD-OF-NDB-ELEMENT	RELATIVE-POSITION *
NDB-ELEMENT-STRUCTURE-OF- NDB-ELEMENT	RELATIVE-POSITION *
NDB-ELEMENT-ARRAY-OF- NDB-ELEMENT	ARRAY-SIZE
IDB-RECORD-TYPE-STRUCTURE-OF- IDB-ELEMENT	RELATIVE-POSITION *
HDB-VIEW-TREE-OF-HDB-RECORD-TYPE	RECORD-LEVEL RELATIVE-POSITION
HDB-RECORD-TYPE-STRUCTURE-OF- HDB-ELEMENT	RELATIVE-POSITION *

The final group defines those relationship-types which are used to define the format of data-elements. In these cases, the attribute-types are dependent upon the second-entity-type in the relationship-type. In the following tables, each relationship-type on the left is associated with all the attribute-types on the right.

Relationship-type	Associated Attribute-type(s)
CDR-ELEMENT-REPRESENTED-AS-INTEGER	LENGTH
CDR-ELEMENT-REPRESENTED-AS-INTEGER	USAGE
CDR-ELEMENT-REPRESENTED-AS-INTEGER	
CDR-ELEMENT-REPRESENTED-AS-INTEGER	
CDR-ELEMENT-REPRESENTED-AS-INTEGER	

Relationship-type	Associated Attribute-type(s)
CDR-ELEMENT-REPRESENTED-AS-DECIMAL	LENGTH
NDB-ELEMENT-REPRESENTED-AS-DECIMAL	SCALE
RDB-ELEMENT-REPRESENTED-AS-DECIMAL	PRECISION
HDB-ELEMENT-REPRESENTED-AS-DECIMAL	USAGE
IDB-ELEMENT-REPRESENTED-AS-DECIMAL	

Relationship-type	Associated Attribute-type(s)
CDR-ELEMENT-REPRESENTED-AS-FLOAT	PRECISION
NDB-ELEMENT-REPRESENTED-AS-FLOAT	USAGE
RDB-ELEMENT-REPRESENTED-AS-FLOAT	
HDB-ELEMENT-REPRESENTED-AS-FLOAT	
IDB-ELEMENT-REPRESENTED-AS-FLOAT	

Relationship-type	Associated Attribute-type(s)
CDR-ELEMENT-REPRESENTED-AS-CHAR	LENGTH
NDB-ELEMENT-REPRESENTED-AS-CHAR	USAGE
RDB-ELEMENT-REPRESENTED-AS-CHAR	
HDB-ELEMENT-REPRESENTED-AS-CHAR	
IDB-ELEMENT-REPRESENTED-AS-CHAR	
CDR-ELEMENT-REPRESENTED-AS-VARCHAR	
NDB-ELEMENT-REPRESENTED-AS-VARCHAR	
RDB-ELEMENT-REPRESENTED-AS-VARCHAR	
HDB-ELEMENT-REPRESENTED-AS-VARCHAR	
IDB-ELEMENT-REPRESENTED-AS-VARCHAR	

7. ATTRIBUTE-GROUP-TYPES

The IDN IRDS Schema has two attribute group-types:

ALLOWABLE-RANGE
DICTIONARY-PERMISSIONS.

Both are as defined in the dpANS IRDS. The DICTIONARY-PERMISSIONS attribute-group-type is associated with the DICTIONARY-VIEW entity-type. The ALLOWABLE-RANGE attribute-group-type is associated with the VALUE-SET entity-type.

8. IRDS DICTIONARY-VIEWS

The following matrix identifies the Entity-Types usage by processor role. The "IRDS-DICTIONARY-VIEW" corresponding to each processor role consists of:

- All Entity-Types as indicated in corresponding column in the matrix;
- All Relationship-Types which associate Entity-types within the "IRDS-DICTIONARY-VIEW";
- All Attribute-Types and Attribute-Group-Types which are associated with either an Entity-Type or Relationship-Type within the "IRDS-DICTIONARY-VIEW"; and
- All component Attribute-types of any Attribute-Group-Type within the "IRDS-DICTIONARY-VIEW".

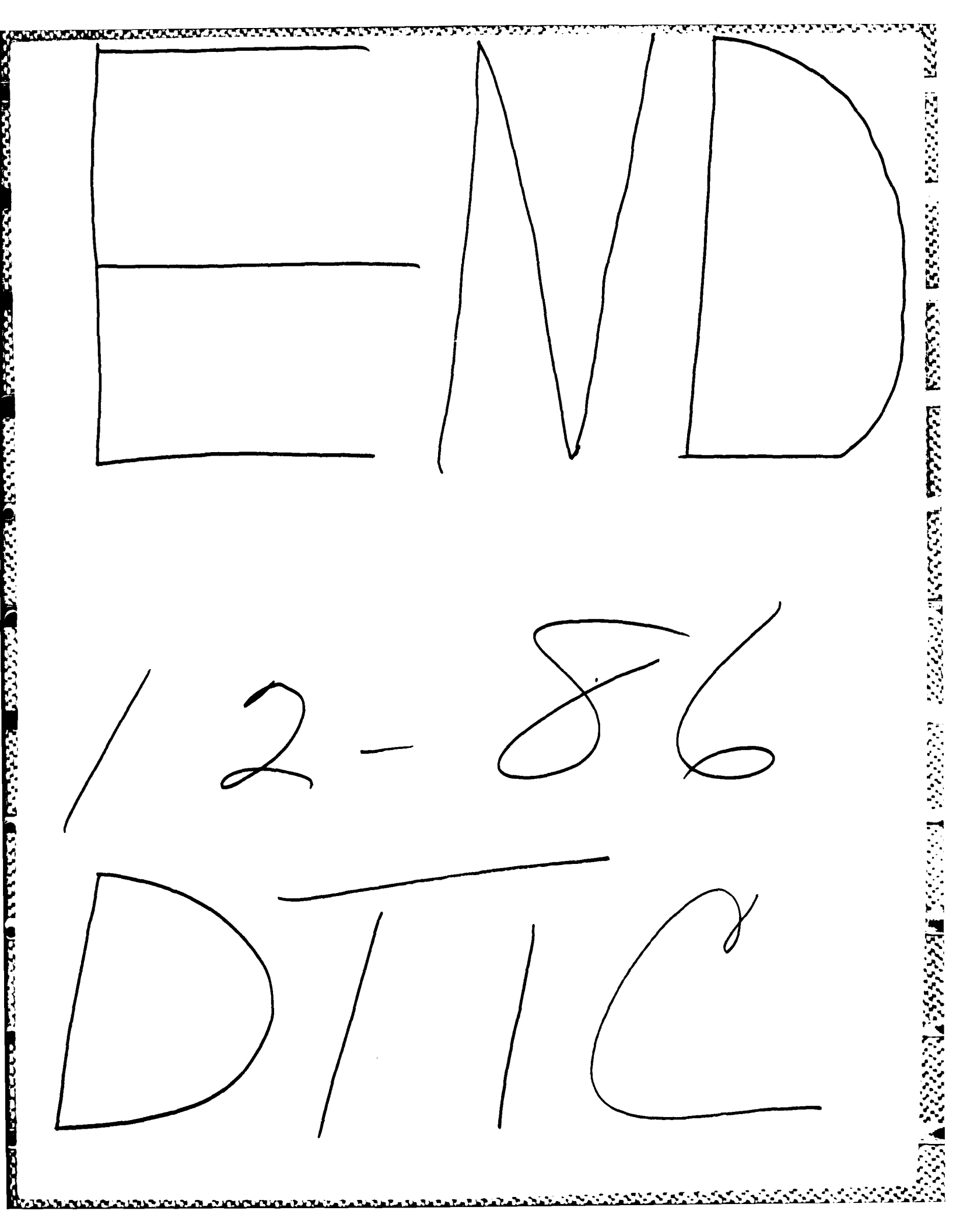
If a processor fulfills multiple roles, its corresponding "IRDS-DICTIONARY-VIEW" is defined by the union of all Entity-types for each of the assigned processor roles, and all Relationship-types, Attribute-types and Attribute-Group-Types as defined above.

In the matrix, an "*" indicates that the Entity-Type is used and consequently must be present in the schema at the node.

	D1	D2Q	D1*
DICTIONARY-USER	*	*	*
DICTIONARY-VIEW	*	*	*
IDN-NODE	*	*	*
STORED-QUERY		*	
ACTIVE-QUERY		*	*
FUNCTION	*	*	*
LOCAL-ELEMENT-NAME	*		
CDR		*	*
CDR-RECORD-TYPE		*	*
CDR-KEY		*	
CDR-ELEMENT	*	*	*
VALUE-SET		*	
NDB-VIEW			*
NDB-SET			*
NDB-ELEMENT			*
RDB-VIEW			*
RDB-RECORD-TYPE			*
RDB-ELEMENT			*
HDB-VIEW			*
HDB-RECORD-TYPE			*
HDB-ELEMENT			*
IDB-VIEW			*
IDB-RECORD-TYPE			*
IDB-ELEMENT			*
INTEGER	*	*	*
DECIMAL	*	*	*
FLOAT	*	*	*
CHAR	*	*	*
VARCHAR	*	*	*

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.


END
12-86
DTIC