

**KNOWLEDGE PROCESSING FOR A
CONSTRUCTION MANAGEMENT DATABASE**

by
MING-TEH WANG

M.S. in Civil Engineering, Rensselaer Polytechnic Inst. (1985)
M.S. in Civil Engineering, National Taiwan University (1978)
B.S. in Civil Engineering, National Taiwan University (1976)

Submitted to the Department of Civil Engineering
in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

20 December 1988

(c) Massachusetts Institute of Technology 1988

Signature of Author _____

Department of Civil Engineering

December, 1988

Certified by _____

Robert D. Logcher

Professor, Department of Civil Engineering

Thesis Supervisor

Accepted by _____

Professor Ole S. Madsen

Chairman, Departmental Graduate Committee

Knowledge Processing For A Construction Management Database

by
Ming-Teh Wang

Submitted to the Department of Civil Engineering
on 20 December 1988, in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy
in Construction Engineering and Management

ABSTRACT

The objectives of project control systems go beyond documentation to recognition of problems and evaluation of their causes. Through the use of broad information systems, a construction manager today is able to assemble a wide variety of data about his projects or programs. Retrieval, analysis and interpretation of the meaning of these data, however, usually requires the user to have detailed knowledge about the structure and content of the database, use of a data retrieval language, and have further programming skills to have the computer perform analyses.

The work reported in this thesis has three goals: to ease retrieval with near-natural language query capabilities, to acquire and accumulate user defined knowledge, and to exercise such knowledge for translation from English terms to database fields or their values, access to database or other information sources, or pre-access and post-access processing. A knowledge based query system, called Expert-MCA, has been developed by coupling the techniques of natural language processing, knowledge-based expert systems, and database management systems. The system architecture will be presented, followed by a description of how knowledge is represented and processed in answering English-like queries. An object-oriented formalism is applied to represent knowledge required in the process of reasoning. Expert-MCA's reasoning capability has been extended by using context driven reasoning, a generalization of reasoning mechanisms used in framed-based systems.

Thesis Supervisor: Dr. Robert D. Logcher

Title: Professor of Civil Engineering

Table of Contents

Table of Contents	3
List of Figures	6
List of Tables	7
1. Introduction	8
1.1 Background Subjects	8
1.2 Research Background	11
1.3 Goals of the Study	13
1.4 Organization of the Thesis	14
2. Related Work	16
2.1 Database Management Systems and Construction Management	16
2.1.1 Introduction to Database Management Systems	16
2.1.2 Use of Construction Management Databases	18
2.1.3 Adding New Information Techniques to Construction Management	19
2.2 Natural Language Processing	20
2.2.1 Introduction to Natural Language Processing	20
2.2.2 Issues in Natural Language Processing for the Construction Domain	22
2.3 Knowledge-Based Expert Systems	24
2.3.1 Introduction to Knowledge-Based Expert Systems	24
2.3.2 Major Issues in Knowledge-Based Expert Systems	25
2.4 Interaction among NLP, DBMS and KBES	27
2.4.1 Natural Language Processing and Database Management Systems	27
2.4.2 Natural Language Processing and Knowledge-Based Expert Systems	28
2.4.3 Database Management Systems and Knowledge-Based Expert Systems	28
2.4.4 Integration of NLP, DBMS and KBES	31
3. Overview of Expert-MCA	33
3.1 Desired System Features	35
3.2 General Architecture	35
3.3 Knowledge Processing in Expert-MCA	38
4. Knowledge Representation and Reasoning in the System	42
4.1 Representation Features in Expert-MCA	42
4.2 Classification of Terms Used in the System	44
4.3 Object-Oriented Representation	48
4.4 Context Driven Reasoning	51
4.4.1 Simple Reasoning Transitivity	54
4.4.2 Forward-Transit and Backward-Transit Reasoning Transitivity	55
5. Language Analyzer	60
5.1 Design of the Language Analyzer	60
5.2 Lexical Mapping	62
5.3 Lexical Analysis	67
5.4 Syntactic Analysis	71
5.5 Semantic Analysis	75

6. Knowledge Reasoner	82
6.1 Structure of the Blackboard in Expert-MCA	82
6.2 Construct of the Reasoner	83
6.3 Use of Context for Solving Query Ambiguity	84
6.3.1 Finding Context	84
6.3.2 Use of Context Driven Reasoning to Resolve Ambiguity	86
6.3.3 Using Context From Adjacent Terms	89
6.4 Use of User and Domain Profile Information	93
6.5 Execution of Tasks Triggered by Terms	96
6.5.1 Rule Base Inference Used for Answering a Query	97
6.5.2 Procedural Processing Used for Answering a Query	99
6.6 Pre-Access Processing and Post-Access Processing	102
7. Query Language Generator and Knowledge Acquisition	106
7.1 Query Language Generator	106
7.1.1 Features of the Query Language FOCUS	106
7.1.2 Composition of FOCUS Code in Expert-MCA	109
7.2 Knowledge Acquisition and Sharing in the System	114
7.2.1 Knowledge Acquisition in Expert-MCA	114
7.2.2 Knowledge Sharing in Expert-MCA	116
8. Implementation Details	119
8.1 Principles and Modules in the Programming Design	119
8.1.1 Programming Design Principles	119
8.1.2 What an Interface Task Does	120
8.1.3 How Interface Tasks Flow from One to Another	120
8.1.4 How to Define an Interface Task	121
8.2 Flow of Modules in Expert-MCA	126
9. Conclusion and Recommendation for Further Study	130
9.1 Conclusion	130
9.2 Discussion	131
9.3 Contributions	135
9.4 Recommendation for Further Study	136
Appendix A. Problems in Natural Language and Processing Issues	138
A.1 Problems Inherent in Natural Language	138
A.2 Major Issues in Natural Language Processing	140
Appendix B. Specification of Terms Used in Expert-MCA	146
B.1 Specification of Fieldname	146
B.2 Specification of Fieldvalue	147
B.3 Specification of Synonym	148
B.4 Specification of Word/Phrase	148
B.5 Specification of Rule	151
B.6 Specification of Procedure	151
Appendix C. Syntax for FOCUS Query Language	153
C.1 Syntax for TABLE Command Statements	153
C.2 Syntax for DEFINE Command Statements	154
Appendix D. List of Interface Tasks in Expert-MCA	156

Appendix E. List of File Names and File Paths in Expert-MCA

List of Figures

Figure 2-1: Coupling between DBMS and KBES	29
Figure 3-1: System Architecture of Expert-MCA	34
Figure 4-1: Abstraction Layers between Users and Computers in Expert-MCA	43
Figure 4-2: Simple Reasoning Transitivity	53
Figure 4-3: Forward-Transit Reasoning Transitivity	56
Figure 4-4: Backward-Transit Reasoning Transitivity	58
Figure 6-1: An Application of Context Driven Reasoning	87
Figure 7-1: A Syntactic Tree for the Definition Slot	117
Figure 8-1: Interface Task Flow in Expert-MCA, Part 1	128
Figure 8-2: Interface Task Flow in Expert-MCA, Part 2	129

List of Tables

Table 9-I: Comparing Expert-MCA with Related Systems

134

Chapter 1

Introduction

This chapter first briefly introduces problems inherent in the construction industry. Next, some technologies which are developed in other areas and might be useful to solve these problems are mentioned, specifically management information systems and artificial intelligence. Research background is then presented, followed by a description about the goals of this research. Finally, organization of this thesis is described.

1.1 Background Subjects

The construction industry has been identified with the following deleterious characteristics: fragmentation, transient nature, non-repetitive process, ineffectiveness of learning, and inefficiency of communication [77, 86]. Today such disadvantages are further compounded by the ever increasing complexity of construction projects that require more specialized equipments and skills, that suffer stricter regulations and more pressures from financial organizations [24]. As a result, the delivery of construction projects tends to involve more participants and more diverse materials and skills. This implies that construction management needs more sophisticated knowledge, more powerful tools, and more advanced technologies than ever before to manage the process of construction projects.

The process usually involves a variety of parties, including owners, design architects/engineers, project managers, superintendents, clerical staff, building departments, material suppliers, contractors, and labors, each contributing to projects with different skills, tasks and objectives. In order to execute their tasks, some parties require expertise which needs vigorous training and long experience, while the others need diversified knowledge backgrounds. Nonetheless, these parties often have quite different motivations in their part of the delivery of construction projects.

All the parties must communicate with each other to make sure that the overall performance of projects meets common interests and fulfill expected functionality requirements. The communication process among parties can be viewed as an information flow. In a construction project, the owner describes to designers what functionality and

constraints the project might have. Typical project constraints include budgeted amount and construction duration available. Such a description actually is a piece of information that serves as an initial set of design specifications. After the description is transferred to designers, a conceptual design and initial estimate of cost and duration are created and then sent back to the owner for review or correction. Based on the information from each other, the owner and designers therefore can refine their decision by modifying old information or create a new set of information in response. Along the process in construction, project managers may manage to control the project's progress by making various decisions, such as organizing field forces, assigning backup personnel in administrative and engineering positions necessary for supervising labor, awarding subcontractors, purchasing materials, keeping records, and engaging in a financial management. Construction information therefore flows from one party to another in a cooperative network.

Computers have been used to cope with enormous amounts of data and to solve problems in almost every business area. In order to save labor cost, increase productivity, or improve product quality, most engineering and construction organizations in this country have gradually made use of management information systems to help handle the data that covers a variety of organizational purposes, such as payroll, job cost accounting, design analysis, and project planning and scheduling. Currently, management information systems that are used for construction management use database management systems and simple applications built around them. Such systems are able to provide facilities for easy collection of broad classes of data and provide some tools for retrieving and reporting such data. Reports are generally direct presentations of collected data with simple arithmetic operations among related items. Reports may select data based on arithmetic or logical conditions among data items, i.e., exception reports.

Management information systems (MIS) can serve as an information exchange center for the information flow that occurs along the process of construction projects. Data captured in a management information system typically is a record about objects in progress, status of milestone events, or decisions which are made by persons with various responsibilities. It is stored in the system encoded to the object. Thus the use of MIS can extract related data about a series of objects from a database and recognize or reason about what happens to the projects. However, the data collected in reports can also be garbage to those who do not know how to interpret it or use it. Therefore real value of data collected in a database can be realized only when users can understand the meaning of data.

For the past few decades, research in artificial intelligence (AI) has been evolving primarily within computer science groups. Currently AI research has broadened to include not only computer science, but also research in areas such as humanity, engineering, and management. Many researchers in AI try to model their computer programs after human intelligence [73]. For example, in order to better use computers to solve the problems of interest, engineers try to simulate the intelligence that they observe in human beings. The researchers with humanity backgrounds may take AI programs as readily available experiments for testing hypotheses about human cognition. In its interplay with other cognitive science disciplines, AI provides an alternative methodology for exploring and testing theories about human intelligence, such as the way we learn, speak, see, or reason.

AI research can be categorized into two types: theoretical research and application research. Theoretical research in AI includes such subjects as search, knowledge representation, knowledge reasoning, learning, and programming languages [6, 7, 8, 88]; where knowledge representation and knowledge reasoning are the most fundamental issues in current AI research. Knowledge reasoning usually covers such topics as deduction and induction. AI application research currently can be further divided into the following groups: knowledge-based expert systems, natural language processing, robotics, speech recognition, and image processing [92].

When facing newly available and advanced technologies developed in AI, we, as members of the construction community, must ask ourselves: What technologies can we adopt from AI research? What else should we do by ourselves in order to fully make use of such technologies? While such issues will be examined in more detail in subsequent chapters, the next paragraph exemplifies how AI technologies have been employed or their use encouraged by researchers in the construction community.

Knowledge-based expert systems might be the most dominant AI technology employed in the construction domain. For example, project planning, project scheduling and control, and other project management issues have been active subjects of expert system development [45, 46, 59]. In this country most knowledge-based expert systems in the construction domain have been developed by academic construction management groups [3, 59]. Examples of such expert systems are: construction planning and scheduling at the Massachusetts Institute of Technology [13, 18, 19, 20, 34, 40], PLATFORM (for construction control, planning, and objective-setting) and SITEPLAN (for layout of temporary construction facilities) at Stanford University [51, 76], CONSTRUCTION

PLANEX (for project planning) at Carnegie Mellon University [33], construction schedule analysis at the University of Illinois at Champaign-Urbana [23], and IRIS (for construction risk identification) at the University of Texas at Austin [4]. Stone and Webster Engineering Corporation of Boston has developed expert systems as advisors for welding and construction management [27]. Still, some others are doing research on the robotics for construction automation [71, 62, 86].

1.2 Research Background

Decision making in construction management is a complex process that requires not only factual data but also heuristic knowledge for judgment. The complexity can be well illustrated by the programming process of the US Army construction projects. In order to achieve their goals, commanders at different levels in the Army need various facilities, such as training fields, barracks, offices, housing, and hospitals. In addition to Congress, army agencies with various responsibilities and authorities make judgments on scope, cost and timing within the lengthy construction programming process, which is called the MCA (short for Military Construction, Army) cycle. The MCA program is a large and seemingly complex system in which projects proceed according to complicated regulations and practices, each taking 8 to 15 years from identification of needs, inception, design, budget, appropriation, to construction. To reach completion each project has to be described, justified, reviewed, revised, programmed, and budgeted by offices and agencies involved in the program development [80]. Projects are ultimately questioned at the Congressional level before funds are approved for disbursement.

The Office of Chief of Engineers (OCE) is ultimately in charge of the management of the US Army construction projects. The Construction Engineering Research Laboratory (CERL) of the Army of Corps of Engineering is a supportive research institute, aiming at providing ways to improve development of Army projects. CERL has been one of the major research organizations on construction management in this country, mostly helping OCE and other Army agencies find better ways to solve their problems.

One of the tools for the management of the Army construction projects is database management systems. Information about the MCA program is maintained in a large, central database containing more than 15,000 projects, with over 500 data items per project, supported by an information system called CAPCES, short for Construction Appropriations

Programming Control and Execution System. The data is used for program planning, budgeting, appropriations management and construction of U.S. Army projects [53, 80].

Use of the CAPCES system has some unique characteristics. Although it is a centralized database system, running on a mainframe in Dallas, Texas, its use (or access) has been highly distributed, meaning that users are located over this country and overseas; users have diverse knowledge background and work in various agencies or groups; data items retrieved by different users can be seldom overlapped. Using personal computer as a handy tool, its users have largely moved toward decentralized use of the information. In order to interact with the CAPCES database, the users, however, must be knowledgeable about CAPCES and a formal query language, FOCUS. Although the CAPCES database system offers many advantages for the efficient collection and reporting of information about the U.S. Army construction program, the real value of the database will be realized only when the users are able to get timely information easily and, more importantly, to understand the meaning of this information and use it to reach conclusions and make decisions.

Specifically, issues about use of the CAPCES database for helping process the MCA program can be divided into two categories: organizational and technical. Due to military organizational practices, knowledge acquisition/accumulation and training for the MCA program may exist some problems. For each project significant decisions have to be made by different people ranging from users of facilities, facility engineers, district, division, major commands, OCE, head quarter of the Department of the Army, to Congress, with different interest such as cost estimate, timing control, funding approval, fund allocation, project bidding, site construction, or facility occupancy. They must know the tasks in the MCA cycle and what information is required for each task. This learning is time-consuming, since the knowledge is widely scattered across different agencies and embedded in voluminous regulations. In addition, military personell are regularly rotated, staying in one position no more than three years.

Involved in the technical issues are access to data, understanding the meaning of data, and use of data. As projects develop in the MCA cycle, information is stored in the CAPCES database and then used by the Army construction staff in the decision making about program. To retrieve data from the database, however, users not only have to be able to program in a query language and communicate through a data network for retrieval, but also have to be aware of the data structure and the meaning of each data item in the database. As can be imagined, it will be difficult to ask the decision makers involved in the MCA cycle to have

broad knowledge of both on computer science and MCA processing. In addition, due to the large amount of data items stored in the database, it is very difficult for users to clearly understand the meaning of each data item¹, not to mention being able to derive or process new information (or new data items).

While programmers can retrieve data for the users who are not familiar with the access to the database, the retrieved data is not always sufficient. Its value is achieved only after post-processing, when the user studies the data, looking for relationships and patterns based on their experience or expertise from sophisticated colleagues. Users often have only a "feel" of what data they need and how to look for patterns in the retrieved data. A knowledge gap exists therefore between these two groups of people.

In construction management, having handy report generators for data retrieval is only a tool for the first step in good decision making. The more important and challenging issues are how one can derive and accumulate knowledge from the data collected in databases, and how one can transfer such knowledge to another. After all, the difference between a sophisticated construction manager and an ordinary one is not because the former knows more about how to retrieve data or generate reports. Rather, the former knows more about the interpretation of data items, the possible patterns among data items, the meaning of relationships of some data instances to other data instances, and use of the data.

1.3 Goals of the Study

While there are still a number of ways that computers can be used better for solving our routine works, it is believed that the issue of using the computer to work smarter is more interesting and challenging. Computers can work more intelligently in construction management, sharing more work that was originally done by humans.

This study has been funded by CERL for developing better tools to manage the process of Army construction projects. A prototype system, called Expert-MCA, developed in this study aims to better solve the problems as described in the last section.

Some constraints for this development are identified as follows. It should run on IBM/AT PCs or compatibles. Within PC environment at fields, users like to have more

¹It often needs to check a table for the meaning of a field and its values, since the database contains enormous amounts of fields and their values.

control on data use and thus operate information on their PCs, a tendency of decentralization of information operation. Since a variety of users do not have sufficient knowledge about data communication, the system should automatically communicate with the mainframe for data retrieval, given a user's password and necessary communication parameters for entering the mainframe. As a result, the system is implemented in a single programming language (i.e., Lisp) which can have overall control of every module.

To most of the staff involved in the military construction management, querying data by using natural language can be a lot easier than using a computer language. Use of near natural language queries seems a better choice as far as the acceptance by most users is concerned. For users to customize their needs and use of language, use of user defined language and knowledge is also required.

The goals of this study, being accomplished via the development of Expert-MCA, are to provide tools for access to database, medium for understanding meaning of data, and use of knowledge defined by a range of users. User defined knowledge is used for translation from English terms to database fields or ranges of values, access to database or other information sources, pre-access and post-access processing, or definition of user's characteristics to establish necessary context.

1.4 Organization of the Thesis

The contents of subsequent chapters are briefly described as follows:

Chapter 2 discusses the subjects that are essential to the development of this thesis, including natural language processing, database management systems, and knowledge-based expert systems.

Chapter 3 depicts the system architecture of Expert-MCA, together with a brief description of how its modules work.

Chapter 4 presents the details of the knowledge representation and reasoning mechanisms used in Expert-MCA. The chapter describes an object formalism that is suitable to the construction domain. The terms, being used to communicate with Expert-MCA are defined by users to convey meanings. The reasoning mechanisms employed in Expert-MCA is also detailed in this chapter.

The major knowledge processing modules in the system are described in Chapters 5 to

7. Chapter 5 describes the language analyzer, being detailed in such submodules as lexical mapping, lexical analysis, syntactic analysis, and semantic analysis.

Chapter 6 describes how the knowledge reasoner resolves ambiguities left from the language analyzer by using information about context, the domain, and the user; execute tasks requiring rule based inference or procedural processing; ask the query generator to produce query code; or activate communication module for data transmission.

Chapter 7 describes the query language generator, which is used to generate FOCUS code for data retrieval. Knowledge acquisition in Expert-MCA is also included at the end of this chapter.

Chapter 8 details the programming implementation of Expert-MCA. The requirements of hardware and software for the implementation are first presented, followed by a set of desired programming features for the implementation. Design algorithms for connecting functional modules are then presented.

Chapter 9 concludes this thesis with a summary of Expert-MCA. The chapter also outlines the contributions of this thesis and recommends further future work.

The reader who is only interested in what this study covers, Chapters 1 and 9 are reading materials to start with. With reading additional materials in chapters 3 and 9, the reader may get brief ideas of how the system works. The reader who goes over Chapters 2, 5, 6, and 7 will get more concrete ideas of how this study has been done technically. If you are interested in how the system is implemented, Chapter 8 and Appendices B and D are a necessity. For those who are new to the subject of AI or MIS, Chapter 2 and Appendix A are good supplemental materials.

Chapter 2

Related Work

This chapter first reviews use of construction management databases. Having discussed the problems in their use, this chapter continues to examine potential advantages of using techniques developed in AI and MIS research to solve the problems. Research work essential to the development of Expert-MCA is then identified and characterized, including natural language processing, database management systems, and knowledge-based expert systems.

2.1 Database Management Systems and Construction Management

2.1.1 Introduction to Database Management Systems

A database is a collection of related factual data for an entity ranging from a person to a big organization. Database Management Systems (DBMS), using electronic devices such as computers, are systems that are built to manage massive databases effectively and efficiently. Such systems all provide tools for facilitating data processing, which may include data collection, data storage, and data distribution. Database systems often encounter problems when the users are getting more and data items/records become larger, or the operation of data processing is getting more complex. Such problems are closely related to such issues as reducing redundancy, avoiding inconsistency, sharing data, data security, data integrity, and data independence [22].

In addition to shielding users from knowing the details of how to physically store/update data in computers, a database management system can provide end users with following benefits: availability of getting accurate and up-to-date data, efficient and easy retrieval.

Data model is a representation of data collected in a database system. It is used as a guidance of organizing and structuring the data in a uniform formalism. The selection of a data model has to consider the following issues: What are the general patterns of data being collected? What data models are available? How is the data going to be used?

Most commercially existing database management systems have been implemented with the following data models: hierarchical, network, and relational [22]. The three data models, however, all have the problems of addressing their data semantically. Namely, they all lack the ability to embody the meaning of stored data. Rather, the interpretation of data items or the relationships between data items is solely depending on users' knowledge on the data items. The importance of understanding data is two-fold: one is on operational issue; the other is on the use of retrieved data. If users are not doing right during an operation, the system may have no way to detect or even to recover the errors made accidentally. It is simply a piece of garbage to users if they do not understand the meaning of retrieved data. As a result, to incorporate more meaning into database systems becomes the theme of semantic data models [37, 63].

The research on semantic data models at present is very active, particularly after its interwinding with the research in AI. In a semantic model, the concepts of entity, property, and association are first identified, followed by a representation as symbolic objects for such concepts. Integrity rules can be applied to these symbolic objects for ensuring the semantics existing in data items. These objects can also be manipulated through a set of operators. Since the early and most influential one of its kind was proposed in Chen's Entity/Relationship model in the mid 1970's [17], various semantic data models have been proposed [37, 63].

As can be imagined, capturing the meaning of the data is a never-ending task, and we can expect to see continuing development in this area as our understanding continues to evolve [22]. For long existing database systems, such as the CAPCES system, it is very unlikely to remodel their data models to capture more meaning of data. Instead, their coupling with new knowledge processing systems is a more promising alternative. In fact, one of the objectives for the development of Expert-MCA is parallel to such a recognition.

Most of data models mentioned above are coupled with data definition language (DDL) and data manipulation language (DML) for users to interact with database management systems. DDL is used for the definition or description of database objects such as: record types, relations, views, while DML for supporting the manipulation or processing of those objects. The language used for retrieving data is also called query language.

2.1.2 Use of Construction Management Databases

Use of database management systems has become a prevailing necessity for helping manage organizational business events such as inventory, payroll, personnel, and accounting. Engineers also recognize that database management systems can be efficient tools for extending use of the data among many participants for engineering applications. Applications of DBMS in business and engineering can be quite different in many aspects, such as how real world objects are identified, how information of the objects are encoded into data and stored in databases, how the data is used, how the data is entered, and how often the data is updated. It has been a challenging issue for managers to manage to better integrate the use of DBMS. This issue is particularly complex to construction managers, since construction management is involved with both business problems and engineering problems.

A knowledge gap always exists between decision makers and programmers in using a large database management system, such as the CAPCES database. Decision makers need information for making their decision. What information they need is depending on how they view the real world and how they would solve their problems. Although they own working domain's knowledge, they may not be good at retrieving data from databases. On the contrary, programmers know how to retrieve data from databases very well. However, they may not realize what data is needed for making decision and how the data is used to solve problems in the application domains.

While most currently available databases can bring construction management a great help in data storage and retrieval, this thesis places a focus beyond data storage and retrieval. Rather the focus is on how the data can be used effectively and efficiently, how information can be extracted from the data, and how knowledge can be accumulated from data processing.

Data is simply garbage if users do not know how to interpret it. Data becomes useful information only when it is meaningful to users. Information about objects may be compiled as knowledge when users familiarize the meaning of the information and recognize the possible relationships or patterns among the data items about objects. The real value of data is not realized until it is associated with or compiled into knowledge that can be used repeatedly in construction management. When the scale of databases becomes larger, the use of data and its interpretation becomes vaguer and more difficult.

A construction management database is not limited to the use as an efficient device for data storage and data retrieval. It can also be a rich source from which one can gather experiences and compile them into knowledge. Since the information about construction management is highly complex in terms of its structure and format, to capture and store the information in an appropriate format is never a trivial task. As can be seen in the CAPCES database, many fields for project status or history of project development store values that are artificially encoded to map with real world events. There exists tremendous information that can be extracted from construction management databases, including the recovery of meaning for the code and other field values. Moreover, ineffectiveness of learning individually at fields and inefficiency of communication among participants, two of disadvantageous characteristics in the construction industry, can be improved by better utilizing databases. Currently, databases might be the only available medium that construction participants can communicate with each other for the data stored. Therefore the data collected in a construction management database deserves more attention for better use. The challenge is how we can better use it.

2.1.3 Adding New Information Techniques to Construction Management

Knowledge processing for problem solving typically consists of the following steps: knowledge formalization, knowledge representation, and reasoning. AI research has been focusing on the subjects of knowledge representation and knowledge reasoning, while leaving knowledge formalization almost untouched. This is because it is impossible for a person to formalize a domain without prior knowledge about the domain in question. This also suggests that AI researchers, most without backgrounds on the construction domain, simply cannot do it for us. Instead, we have to formalize the domain by ourselves in such a way that we could better make use of newly available information technologies to improve our performance.

Information processing technologies such as those used in database management systems and knowledge-based expert systems have been widely recognized as powerful tools for improving work performance in the area of construction management [38]. However, the techniques employed in natural language processing and object-oriented programming are also needed [53]. It is our responsibility, and also a challenge, to adopt such technologies to solve our problems or improve our work performance.

With the complexities and characteristics of construction management in mind, we

have to study how such information processing technologies should be applied to help solve problems in construction management. A key to a success for the application is that we have to know the strengths and the limitations of such techniques, the characteristics of our problems, how such technologies can be fitted into our domain frameworks and then work out desired results.

In order to solve the problems of an interdisciplinary subject such as construction management, one needs more integrated systems, by incorporating various knowledge sources and adding new information processing technologies to better make use of the strengths of individual ones. Specifically, such integrated systems should be coordinated with better information flow procedures in an organization; be allowed to have a broader use of information; and be dynamically customized to meet user's needs [53].

While many research issues are all critical and essential to what and how new information processing technologies can be added to construction management [49, 50, 77], this thesis presents a prototype system trying to integrate some information technologies, being discussed in the following sections, with the following objectives: to achieve a broader use of information from a construction management database or related information sources; to allow users get timely information by using near-natural language queries; to allow users to define knowledge for such processing.

2.2 Natural Language Processing

In this section, natural language processing (NLP) is introduced, followed by a discussion of how conventional NLP can be difficult in dealing with some problems. Examples from the construction domain illustrate a need for a better approach.

The reader who is not familiar with problems inherent in natural language and major issues in natural language processing is strongly referred to the materials in Appendix A.

2.2.1 Introduction to Natural Language Processing

Current development of natural language processing has evolved from research in areas such as linguistics and computer science. Natural language, as opposed to formal languages, which are artificially developed to communicate with computers, has been learned and used for daily communication since our childhood. The process of understanding natural

language and generating further representations which are used to communicate with computers is called **natural language processing**.

In addition to the knowledge about the domain discussed in a communication process, the understanding of natural language needs a variety of linguistic knowledge, such as the awareness of the structure of a sentence, the meaning of each word in the sentence, the implication of context and discourse. In order to allow computers to understand and thus communicate with target participants, many researchers, primarily with computer science backgrounds, have devoted themselves to the issue of how to represent and reason with the knowledge that is conveyed in a communication process.

Primary phases in natural language processing are syntactic analysis, semantic analysis, and discourse interpretation. Natural language processing has to deal with, among other factors, the following issues: how to characterize the capabilities that the system should have, how representation is used for internal processing, how the system architecture is designed, and how the system can deduce or reason new information logically and correctly.

Being fundamental to the other phases, syntactic analysis plays an important role in natural language processing. It deals with how the words in a sentence group into phrases, and in turn how these phrases group into larger phrases. Grammars and parsing algorithms are the core issues in this analysis. A parser is a construct that employs an algorithm to produce a structured representation, called a parse tree or a derivation tree, for a sequence of words according to a set of grammatical rules. Based on the definition of words and phrases, a parser should be able to form efficiently a structured representation that in turn can help interpret the sentence as a whole in semantic analysis. The various constructs of parsers reflect the different views of their designers on how the words are used structurally and semantically in sentences, what problems are to be attacked and what knowledge should be used in order to attack the problems [64].

Although syntactic analysis is currently the most well-developed area in NLP, conventional approaches employed in syntactic analysis are not always satisfactory to parse sentences which consist of words with multiple meanings or multiple word classes. In other cases, sentences may be further compounded by containing **split terms**², each consisting of

²A military engineering officer may request: "What are the projects in fiscal year 88 with program amount over 5000?", where the split term FISCAL YEAR 88 consists of a primary term FISCAL YEAR that expects an associated variable term; in this case it is the number 88.

several words that are separated in position and as a whole contribute to a complete set of meaning. Moreover, many professionals tend to use their domain jargons which may cause words to have different meanings in different contexts.

The following section will identify problems which traditional systems of natural language processing are found inadequate or difficult to resolve, and thus propose a more promising approach.

2.2.2 Issues in Natural Language Processing for the Construction Domain

To capture ideas underlying the literal expression of sentences, participants in a communication process not only need linguistic knowledge, but also information about context and knowledge about the domain in discussion. Owing to the lack of context or knowledge background of a domain, a novice may not capture the meaning of an utterance about that domain at all, even if he or she understands quite well the meaning of each word and the syntactic structure of the utterance. This is the case that often happens to students when they start a new subject.

Based on their observation on language use, researchers have been building a variety of natural language processing systems for the past few decades. Since natural language has a great deal of complexity in its use, researchers often tend to work on a particular part of the full scene that natural language performs. As a result, different systems designed for language understanding (or natural language processing) so far reflect the facts that their designers view language use differently; that some problems are more important than others and should be resolved first; that different application domains are chosen for best illustrating their key arguments about natural language processing.

Currently, many natural language processing systems have been designed to interface with existing databases, using one parsing approach or another depending on their application domain and the language patterns most frequently used in the domain. For example, the INTELLECT system (formerly called ROBOT), one of the few systems that has already been marketed, is designed to work with any database of the right form, using the contents of the database itself as the information source for determining the meaning of its input sentence based on a semantic grammar approach [91, 93]. It can parse input sentences and retrieve data stored in a target database, with a simple calculation to the stored data if necessary.

However, most conventional systems for natural language processing do not provide mechanisms, or their capacity is inadequate, to cope with the following issues: how to specify the contexts involved and the information about users, how to define knowledge for resolving ambiguities, how to specify meaning of terms or data items, and how to further process on retrieved data. These facts reflect the need to design systems that require more knowledge for handling requests that often go beyond the capability of currently available systems. As a result, Expert-MCA is designed to provide utilities for solving the problems as mentioned above.

The need to cope with the above issues can be exemplified by the following queries.

1. What are the FY 88 projects in MA and PA with PA over 900 and construction completion percent less than 80?
2. What are the housing projects that need congressional notification?
3. What are the design funds in FY 88?
4. When is the latest date to submit projects to OCE for FY 91?
5. Show the design performance for FY 87 projects by designers.

Use of language (or meaning of terms) can be domain dependent and context involved. In the first query, the word MA may have two meanings: state code is 25³ and program code is MA (for military army projects). Note also that there are two PA's in the sentence. The first one stands for the state of Pennsylvania and the second one for Program Amount. Semantics is required to differentiate the two. In the second query, "housing" can be defined as "CATCD3⁴ IS 173 OR 175," while "congressional notification" as "(Current Working Estimate - Program Amount) > 2000 OR (Current Working Estimate / Program Amount) > 1.25."

Concept in use of language can be indirect to a field name or field value. The term "construction completion percent" is not directly related to a field name in the CAPCES database. Rather, it shares the field DES_PERCENT with the term "design percent," but with different ways of inserting a leading letter⁵ to its field values. Namely, "construction

³Although MA can also be defined as a term with the definition: "State is Massachusetts", or as a synonym for the term Massachusetts, which is associated with a field STATE, this thesis will stick on the definition: "STATECD IS 25," where STATECD is for state code.

⁴It stands for project category code with 3 digits.

⁵In this case, the letter C is used to combine with a percentage for "construction completion percent," while the letter D is for "design percent."

completion percent less than 80" is encoded as "DES_PERCENT LT C80," while "design percent less than 80" as "DES_PERCENT LT D80."

Answers to queries may not be from individual fields, but from statistical, mathematical, or logical operations on data. The derivation of design funds⁶, as in query 3 above, is a process of combining logical operations on some fields and mathematical calculations on other fields.

Answers to queries may not even come from databases, but from inference with domain knowledge. To find the latest date to submit projects to OCE⁷, as in query 4 above, one has to consult with regulations or know the practices.

To derive answers to queries can also be a very complex process, combining data retrieval, pre-access and post-access procedural processing, and rule based inference. To evaluate design performance of a designer⁸, for example, one has to identify the projects being designed by the designer, evaluate the design performance for each of the projects, and average the design performance over the projects for the designer.

2.3 Knowledge-Based Expert Systems

Knowledge-based expert system (KBES) is a computer program which can serve users as a consulting expert or an assistant in a narrow and specific domain area. While some researchers deliberately distinguish underlying ideas of knowledge-based expert systems, knowledge-based systems, and expert systems [38, 87], this thesis does not try to make distinction among them. Instead, we use the term knowledge-based expert systems to include the other two, or use them interchangeably.

2.3.1 Introduction to Knowledge-Based Expert Systems

Prior to the last decade, AI researchers tended to rely on non-knowledge-guided search techniques or computational algorithms for problem solving. These techniques were very

⁶This is illustrated in Section 6.5.2.

⁷This is also illustrated in Section 6.5.1.

⁸Please see Section 6.6.

useful in solving well-structured problems. However, realistic problems usually have characteristics that require an enormously large search space involving many parameters. For such a problem, these search techniques have generally proved to be inadequate and a new approach is needed. The new approach emphasizes use of knowledge explicitly. For example, knowledge can be used to bound or reduce search space, to guide how to do inference. This new notion of knowledge use has led to the field of knowledge-based expert systems, which are a better alternative for solving problems in the domain that are ill-structured or the conventional programming systems are not suitable.

The unique features of knowledge-based expert systems may outweigh conventional programming systems in solving problems in some domains. Conventional programming, such as the ones written in Fortran or C, often requires the programmers themselves be the experts of the domain for which the programming system is written. Its programming style is procedure-oriented, meaning that a program is a collection of procedures which mix domain knowledge and inference knowledge. The mixed use of domain and inference knowledge in programming code has made it very difficult to understand and maintain.

On the contrary, knowledge-based expert systems separate the knowledge for inference from that about a specific domain. This new programming style facilitates many benefits, such as expertise knowledge or heuristics acquired from experts could be easily implemented; the system can grow by adding/updating new knowledge incrementally; the knowledge encoded is transparent to users or experts so that they can easily understand or debug the reasoning results; the reasoning process can be traced if necessary.

The module that does inference in a knowledge-based expert system is often termed **inference engine**. Typically an inference engine consists of two special sets of knowledge: that serves as rules for inference (such as deductive rules) and that controls how inference is processed (such as forward chaining or backward chaining).

2.3.2 Major Issues in Knowledge-Based Expert Systems

Inference capability and knowledge representation are the two key issues in knowledge-based expert systems. They are closely related. To select a knowledge-based expert systems for an application domain, one has to take into account both of them. Different representation systems can be used to represent a domain knowledge once the domain is formalized. Rules, frames, semantic networks, and formal logic are the representation systems most frequently used in knowledge-based expert systems currently.

Each of the systems tends to be better in one aspect, but may be worse in another. The expression of knowledge and reasoning process of a rule-based system are transparent to all kind of users, but it is not suitable for representing procedural knowledge. Although it can grow by incrementally adding new rules, its performance may decrease dramatically when the number of rules is getting larger. As long as we can have clear semantic specification for the formats and symbols used in a formal logic system, its reasoning capability has proved to be logically correct given factual premises. However, it cannot guarantee to run efficiently and effectively in every domain. Nor is it appropriate for representing procedural knowledge. In many aspects, we may think that frame-based systems were evolved from semantic networks. In general, frame-based systems are quite flexible in expressing both procedural knowledge and declarative knowledge. However, their reasoning capability can differ very much from one system to another.

Expert-MCA has applied the framed-based systems as its backbone for its knowledge representation, together with reasoning mechanisms that combine those embedded in other representation systems. For example, context driven reasoning⁹ is used to identify (or infer) the relationships between objects semantically. To manage information in its processing, Expert-MCA also employs the concept of blackboard, which is introduced in the next two paragraphs.

A blackboard is typically used as a temporary environment or working place for exchanging information among individuals of a group of people, each holding some kind of expertise for solving problems in question. The concept of blackboard has been extracted by AI researchers to model our problem solving process and organize knowledge with hierarchical nature [60, 61, 82]. The blackboard model has been applied to solve various problems, such as the HEARSAY-II speech recognition system [25], SITEPLAN [76], and BB1 [28].

A blackboard framework consists of three major components: knowledge sources, control, and blackboard data structure. To solve a problem, we may need a variety of domain knowledge, which can be partitioned and kept separate as different knowledge sources, each being capable of solving a sub-problem. The various knowledge sources can modify or update the information which is posted onto a so-called blackboard, or a global database. All

⁹This is a generalization of inheritance mechanisms embedded in frame based systems. Please refer to Section 4.4 for more details.

modifications to the information on the blackboard are explicit and visible to the knowledge sources. Control mechanisms are required to monitor changes of the information and the status in solving problems, and hence to decide who is going to work next and when. It does not matter what knowledge representation a knowledge source is encoded. Knowledge sources can be represented, for instance, as rules or procedures. However, it requires uniform syntax and semantic for representing the information on the blackboard.

2.4 Interaction among NLP, DBMS and KBES

The technologies employed in natural language processing, database management systems and knowledge-based expert systems have proved to be useful in various domains. As needs and technologies continue to evolve, however, the emerging of the three becomes prevailing both in academic environments and real world applications. This section discusses how they can emerge.

2.4.1 Natural Language Processing and Database Management Systems

The easiest way for a casual user to communicate with a database management system is via natural language front-end interface. As data in data management systems is getting more complex and the systems become larger and more distributed in use and location, the capability of shielding users from knowing the details of the databases or writing formal query languages seems to be a necessity in the future. With this in mind, researchers both in academy and industry have been geared to develop better NLP techniques for interface with database management systems, mostly with those used currently.

Most application research on the subject of natural language today are working on the issue of how we can design a natural language front-end in which users, both sophisticated and casual, are able to communicate, often for data retrieval only, with database management systems, because database management systems are readily available in enormous organizations and, more importantly, there exists a constant demand for such communication. Issues such as transportability, effectiveness, distribution are important to this emergence. Some of such research results can be exemplified in the following natural language processing systems: TEAM [30], ASK [74], EUFID [70], Ginsparg's system [29], IRUS [9], CHAT-80 [83], and LDC-1 [5].

2.4.2 Natural Language Processing and Knowledge-Based Expert Systems

Knowledge-based expert systems may require natural language processing for generating explanations about their reasoning process and results in English-like statements. In order to respond in a more friendly fashion and explain a reasoning process, knowledge-based expert systems need to provide facilities for users to interact with them in English-like statements. Therefore provision of natural language processing is essential to meet such a need. An early knowledge-based expert system MYCIN, for instance, implemented a module to handle the interface with users in a natural and transparent style [58].

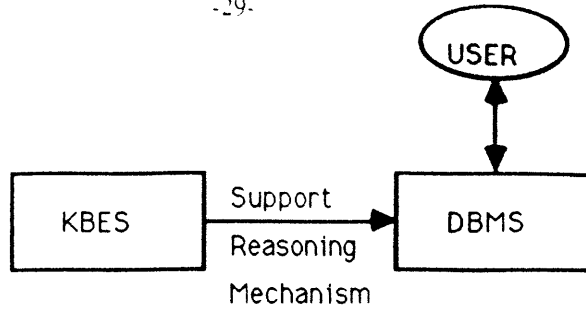
Knowledge-based expert systems could provide explanations about: how a decision is made; how one piece of information is used; why another is not used; why it fails to make a certain decision; what the system is doing now. It is believed that there are reasons for a knowledge-based system to explain its reasoning. These might be for the purpose of understanding, debugging, education, acceptance, and persuasion [69].

The other perspective that we can draw from the relationships between NLP and KBES is following: KBES can serve as knowledge sources and inference mechanisms to help process natural language. Much of the knowledge required to interpret the words or contexts involved in natural language is heuristic and domain dependent. With this in mind, one can incorporate the expertise knowledge about lexicon, syntax, semantic, discourse context, and pragmatic domain into a knowledge-based expert system which in turn provides language analyzer necessary information in its course of processing natural language. While this idea is no longer new, many researchers have devoted themselves to this end continuously [21, 68]. The important works of this line include Schank's conceptual dependency theory [66], Schank and Abelson's SCRIPTS [67], Wilensky's SAM [90], and works by Carbonell, Cullingford, and Gershman [15].

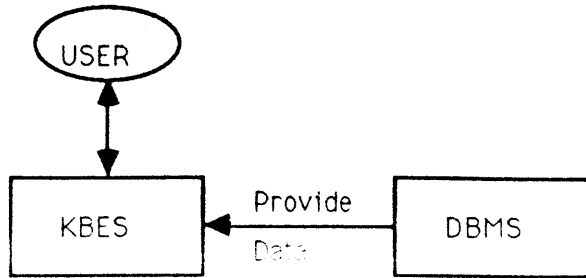
2.4.3 Database Management Systems and Knowledge-Based Expert Systems

DBMS is good in processing large amounts of data, while KBES is good in problem solving. Four types of coupling between these two systems can be identified, as graphically shown in Fig. 2-1. The first one is that knowledge stored in KBES can be used to direct DBMS how to proceed in data processing, such as data updating and data retrieval. In such a coupling, KBES indeed serves as a front-end to DBMS. By doing so, we can add to DBMS with semantics that will, for example, help maintain data integrity in databases. Without the

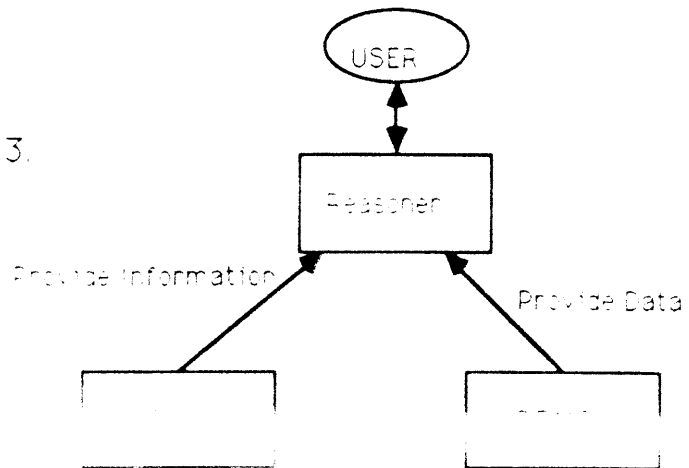
Type 1.



Type 2.



Type 3.



Type 4.

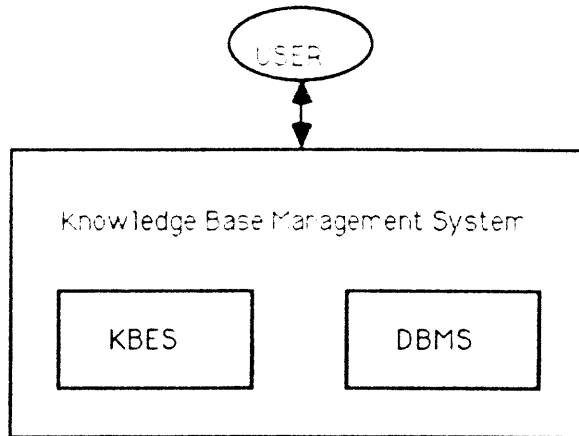


Figure 2-1: Coupling between DBMS and KBES

database system, such kind of coupling is somehow able to be implemented by semantical data models. An intelligent database system developed by GTE, is an example of such coupling system [41]. This is discussed in proceedings on Expert Database Systems [44, 72].

DBMS can serve as a supplemental information source to KBES. Results are useful to its users. If necessary, mapping mechanisms are used to transform expressions in KBES to target query language code, for data retrieval. The data obtained from databases is next to be further processed in KBES. Users of the KBES may not see two systems. The development of KADBASE is a good example [35, 36]. However, its use of rule-based representation adds to the complexity that mapping of expression between KBES and

two systems can in parallel serve as different information sources. A user should sit on top of the two systems and decides which one to use at a time. User's requests for information retrieval can be processed by either of the two systems and eventually merges the information retrieved either from the expert systems into the one that makes sense to users. In this case, the semantics of the information retrieved from databases and from KBES may be quite different. Namely, the former is more about facts which can be measured objectively by any person or can be described by nouns. On the other hand, the latter is more about heuristic knowledge which can be described by verbs. The CIS/TK system (Knowledge Information Systems) falls into this category [55, 84]. As can be seen, this thesis employs this type of coupling in conjunction with the information requested by users.

The concept of knowledge base management systems (not to be confused with all the information currently stored in databases or knowledge, which should be managed in a unified way just like a database management system. Many issues raised by such systems were discussed in a 1985 conference [14].

Several strategies for knowledge-based expert systems and database management systems are suggested: enhancements of existing systems,

resulting in a new class of expert systems. Implementing a sophisticated data access and reasoning capabilities. The coupling between KBES and DBMS employ the KBES functions for more than one time play the role of an expert system on a very large database. This strategy is implemented by the DBMS and to act as an integration of KBES and DBMS is not constrained by KBES and DBMS limitations on both systems.

Efficiency is concerned. Rather, its choice of algorithms to be solved, constraints of hardware power, cost, and time. In the case where performance would be more attractive, since the search cannot interrupt CAPCES's tremendous ongoing projects.

The coupling described above becomes more complex if its from these three and to solve a problem in a certain domain, there exists a great variety of strengths of them can be fully utilized. If solved, the design or arrangement of

Expert-MCA is an integrated system with which all queries are received by a natural language processing module which will decide what system in Expert-MCA can play the role of the knowledge source providing advices to the natural language processing module

and for constructing more complete queries with the knowledge about the user and the domain. In parallel to the CAPCES database system, it can also serve as another information source to the reasoning module by executing rule based inference and procedural processing, which are triggered by some terms in the input queries.

Chapter 3

Overview of Expert-MCA

Just as do database management systems facilitate users the use of data at present, so can knowledge processing systems help the practitioners to organize its knowledge structure, to collect and maintain knowledge systematically, and to infer new facts automatically. Such knowledge processing systems are deliberately integrated by incorporating various knowledge sources and adding newly available information processing technologies, together with comparable reasoning capabilities, to better make use of the strengths of individual ones. Specifically, such integrated systems should be coordinated with better information flow procedures in an organization; be allowed to have a broader use of information; be dynamically customized to meet user's needs; be able to automatically reason new facts [53].

Expert-MCA is a knowledge processing system which provides both knowledge acquisition facilities and problem solving. Its primary application is convenient querying of a large database, but its reasoning capabilities can be used for other problems. Its architecture, shown in part in Fig. 3-1, will be explained in this chapter. The technologies included in the system are natural language processing, rule based inference, procedural processing, database data processing, and object-oriented representation and programming languages.

This chapter first presents the desired features for the system design of Expert-MCA, followed by a brief overview of its system architecture. Next, it describes the processing in the Query Session, a major session that users run in Expert-MCA.

Note that the materials in this chapter give only a brief overview about what is done in each of the modules in the Query Session. How such modules are designed and processed are described in the subsequent chapters. The reader who is interested in how the system is designed and processed in more details may take a look at Fig. 3-1 and then skip this chapter.

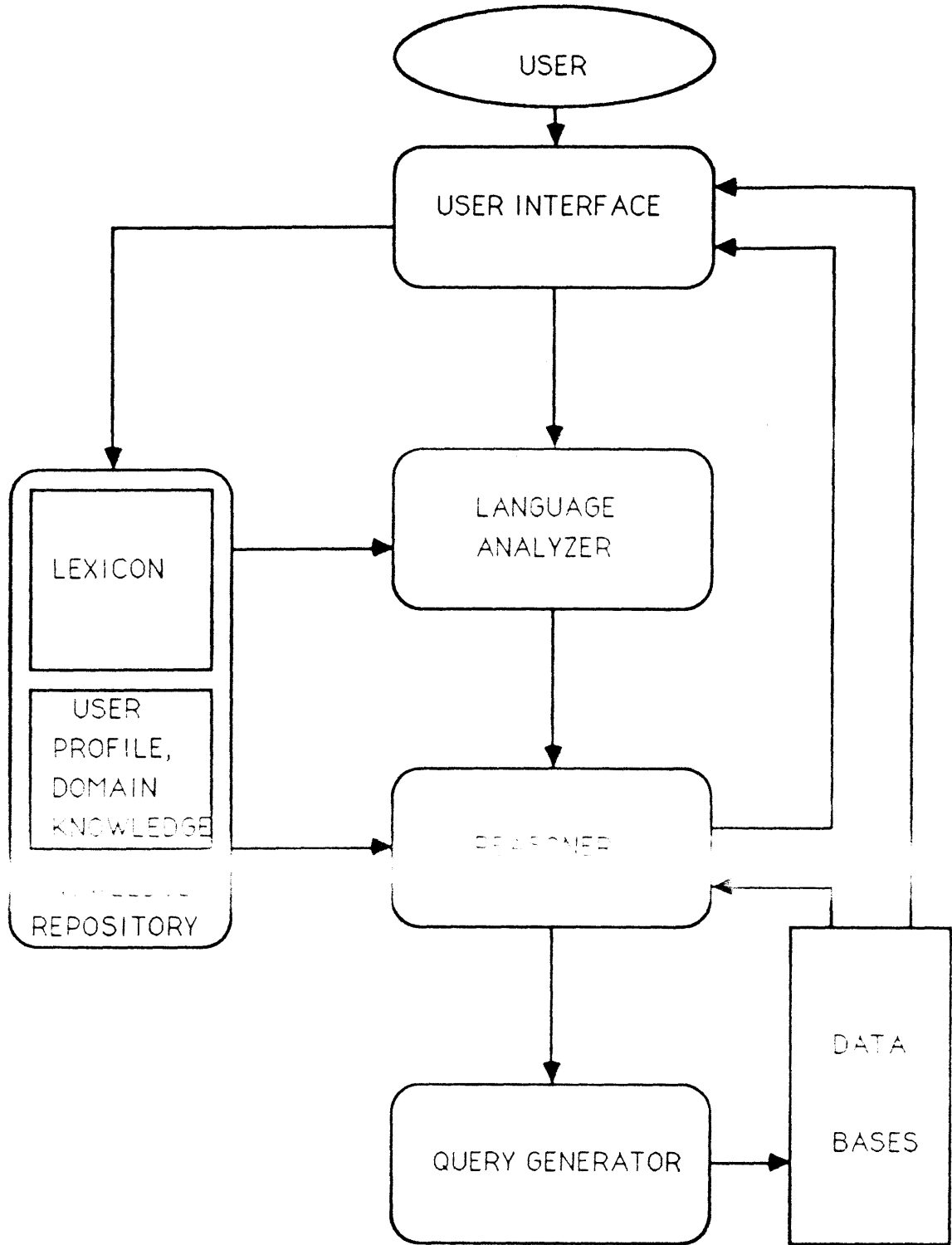


Figure 3-1: System Architecture of Expert-MCA

3.1 Desired System Features

Expert-MCA is designed under the considerations of the following features: friendliness, transparency, expansibility, transportability, and effectiveness. By friendliness we mean that the system emphasizes mechanisms for ease of use such as by providing learning examples, on-line help, detailed help messages as operational guides. The words or phrases used for requesting information should be as transparent to user as possible, since it is users who use and maintain the system. For simplicity, a word or a phrase used in Expert-MCA is called a term. In order to allow users customize their needs, terms can be defined or modified as they need. The definition of terms should be as easy and apparent as possible, either with a free format or a guided approach. The system grows as it accumulates its knowledge by having more terms and incorporating more knowledge sources.

Since ambiguities inherent in natural language sentences cannot be completely resolved, the system allows users to correct its understanding of the query sentences if necessary. In order to be adopted by similar applications, the programming style of the system should be as modular as possible so that code revision and preparation can be as less as possible.

3.2 General Architecture

The system is largely menu driven, with free field data entry into slots within screens. The major session that users run in Expert-MCA is the **Query Session**, which is used to produce answers to near-natural language queries. Its supportive modules include **Teaching Session**, **Profile Update Session**, **Download Session**, **Standard Report Session**, etc. In the **Teaching Session**, users can teach the system with new words and phrases or learn from the system about how terms are defined. Users can update or review user profile and domain profile in the **Profile Session**. The user profile specifies information about user's background information, including the agency and job the user is working on, the particular projects or time span he or she is interested in, default data items for queries, the password and communication parameters for him or her to enter the mainframe in Texas, etc. On the other hand, users can specify knowledge about the domain in the domain profile, which is used to reason new facts necessary for producing answers to the input query. How the profile information is used and entered will be discussed in the subsequent chapters, while rest sessions are presented in Chapters 7 and 8.

The system uses a blackboard architecture where facts are posted to the blackboard by various processes of the system. For example, when a user logs onto Expert-MCA, the date and a variety of information from the profile about the user and domain are posted. At specific points, the reasoner, an inference processor in the system, is called to exercise a specific rule or a procedure against the facts on the blackboard to derive new facts or supply results.

The query capability uses of five major modules: a user interface, a language analyzer, a reasoner, a knowledge repository, and a query generator. The user interface during query is free field input in near English. The user's requests are passed to the language analyzer, which finds word and phrase meanings in the lexicon and transforms the input into an internal representation. The reasoner then works to form and execute a query. It does so by operating on the word and phrase definitions, executing in the process any procedures and inference processes identified within these definitions. If a database query is identified, it will call the query generator to generate query code for retrieving data from the CAPCES database. The query generator then calls the database, which may be either on the mainframe or subsetting on the PC. It can read the PC database structure to determine if all data needed for the response exists on the PC. If not, it provides the user with the option of accessing the mainframe or modifying the query. A communications package provides transparent access to the mainframe and data is then imported into Expert-MCA. The reasoner may operate further on this data or merely pass on the output as a report.

The choice of architecture for the language analyzer (or natural language processor) depends upon the scope of information, the nature of interpretation problems to be resolved, and how much ambiguity exists in the natural language [64, 91]. All the natural language processing systems must build at least an internal representation for the input query. A natural language processing system for database retrieval should have information about words, phrases, syntax, semantics, and a target query language.

The language analyzer in Expert-MCA is a context driven parser which transforms the English query into an internal (or intermediate) representation. This consists of a list of words or phrases in a semantic tree structure based on word or phrase type along with their dictionary definitions. These definitions may include procedures which are sequences of data manipulation and reasoning to be executed as a consequence of the use of the word or phrase. Arguments for the procedure may come from the blackboard or other words or phrases in the query. The reasoner then tries to move the meaning of words or phrases into

query slots, these slots identified as requested results, selectors, and sorters. The system looks for procedures, stored with trigger words, and executes these procedures from left to right in the order of the trigger words. The procedures may take and leave information on the blackboard, may bind other words or phrases in the input to the procedure, may call other rule bases which reason from information on the blackboard, and retrieve and manipulate information from the database.

The knowledge repository consists of a lexicon, various procedures and rule bases, and information about the user. The lexicon is a collection of terms used by users and the system. It also includes the data definitions in the target database. Some of the terms are synonyms for data fields or items, others are synonyms for values or ranges of values for data fields. For example, "family housing" includes a range of values of a field "category". Other terms have semantic meaning or reference procedures.

Knowledge is primarily stored in rule bases and procedures. Basic knowledge is modified and supplemented by the user. Knowledge includes interpretation of the data, user profile, syntactic knowledge, semantic knowledge, and report generating knowledge. The reasoner calls such rule bases or procedures after the parsing by the language analyzer. The blackboard is used as a working area for information updating and exchange during the reasoning process. It consists of various slots for storing all the information needed for Expert-MCA.

The requested information can be filled from terms referencing the database, from procedures, or from execution of a rule base. If requested information does not exist in the database, the user will obtain results from a procedure which will print results on the blackboard. The system is therefore not limited to database queries. However, if an answer to the particular query requires information from the database, the reasoner will activate the query generator to generate a list of the database retrieval commands, which will be displayed to the user, as an option, for his modification. The command list is, then, used to query the target database. The query generator is the module that translates the user's representation to FOCUS commands used in CAPCES. If Expert-MCA is not able to recognize words in the query, the user is asked to extend the system's vocabulary through a series of straightforward interactive screens provided in the user interface. These new words or phrases are stored in the lexicon for future use. In this way the system is personalized for each user.

Many queries can be answered by filling in the query slots for results re

selectors, and sorters, and retrieving the results from the database. The database output then becomes the report. More complex queries, coded in procedures, can import data retrieved from the database and perform further pattern recognition searching among the data.

The system provides several tools for extending, personalizing, and learning to use it. These include a teaching capability and facilities for defining new words and phrases. They also include a capability to input and edit procedures and rule bases. For these, the user is aided by prompts related to the structure of the input. For example, in defining a procedure, the system knows each line must start with a command. The user can easily select among the alternatives. These must be followed by a variable, and so on. Editing existing knowledge is equally easy.

3.3 Knowledge Processing in Expert-MCA

While the details of knowledge representation and reasoning in Expert-MCA are described in Chapter 4, the section only presents an overview about its reasoning process from a functional standpoint. Chapters 5 and 6 have more subtle discussions about such issues as why an approach is employed, how a mechanism is constructed, and how each of the modules is processed.

Reasoning in this knowledge based query system is primarily performed in the language analyzer and the reasoner. It includes analysis of the input query, coordination of information retrieval from the database or rule bases, and manipulation of the retrieved data. In this section knowledge representation used in the system is briefly covered before describing how knowledge processing is done in the system.

A user query is composed of many terms. A term can be a single word or multiple words in structure. The terms used in Expert-MCA are classified into six types: database field name, field value, procedure, rule base, synonym, and other word or phrase. Each term represents an object, a concept, or a knowledge unit related to the domain of the application. A frame representation is used to store the terms used in the system, with slots for different types of information about the term. Terms of the same type have an identical set of slots. The frame name is the term itself. All the terms are collected in the lexicon of the system. Except for terms which define database fields and commonly used vocabulary, terms are defined and maintained by the user. From a database management point of view, the terms act as a data definition language (DDL), while the user's input sentences as a data manipulation language (DML).

The lexicon contains terms which are values of and related to specific data fields in the CAPCES database. For example, Massachusetts is a value for the field named STATE. Field names are also stored, such as FY for fiscal year. A term synonymous with another term can be defined as a synonym for the other term. For example, MASS and MA can be defined as synonyms for Massachusetts. A procedure or a rule base, just like the other terms, is defined by the user. It is associated with a trigger term which can be used in user's input queries or it can be called from another procedure. Usually, a rule base is used to find new facts, while a procedure is used to form a pre-access processing part (for example, definition of temporary fields) for data retrieval, or to manipulate tables of retrieved data.

Word/Phrases are used for terms that are not classified in the other five categories. "In" and "for" are such terms. "Cost overrun" is also a term of type word/phrase. In its meaning slot the cost overrun frame has a slot value "current working estimate greater than program amount", where "current working estimate" is a synonym for a field CWE, "greater than" is a synonym for the other word/phrase "GT", and "program amount" is a synonym for a field PROG_AMT.

After query input, the language analyzer develops an internal representation for the query. Within the language analyzer, there are several phases: lexical mapping, lexical analysis, syntactic analysis and semantic analysis. The lexical mapping module is to retrieve the definition for each of the terms in the input query and push it into an information stack. If any word in the information stack is indexed with "undefined", then the system will ask the user to define the new word and it will process the whole sentence again. The user also has the option to ignore the word or correct it if it were misspelled.

After the lexical mapping is complete, the information stack consists of a sequence of terms and their associated definitions, including syntax and semantics. The lexical analysis module first assigns phrases a higher priority than individual words and longer phrases higher than shorter components in the retrieved definition stack, and second tries to bind words into phrases according to information¹¹ about how to form a new phrase by binding a key term with its neighboring terms.

A database query can be viewed as an attempt to collect a subset of data from a set of

¹¹The needed information for such a binding is not the ones about parts of speech. The term newly formed in such a binding process is called split term. This is to be explained in more details in Section 5.3. Note that the information about parts of speech for terms is needed in the syntactic analysis module, as can be seen in Section 5.4.

data which is stored in a database. For such a collection, it is first necessary to know which data items will be collected, and then to specify possible screening conditions which such data items must meet. How to arrange the data in an output report can also be embodied in a query.

Therefore, the syntactic analysis module tries to segregate, after using syntactic information to form a partial parse tree, the terms in the user query into requested data items, screening conditions, and sorting conditions. A set of heuristic rules is used to group the components of the parse tree into these three parts. For example, terms that relate to possible values of data fields indicate selection criteria. The word "by" or data field names without values indicate sorting criteria.

Semantic analysis, which is next, is a process of capturing the meaning of both terms and the sentence as a whole. In Expert-MCA, semantic rules are used to refine the three parts for determining query constituents for data retrieval. The query constituents, as a result of the semantic analysis, are expressed in terms of an internal representation and posted onto the information blackboard.

In conjunction with the information posted to the blackboard, the reasoner next undertakes several tasks. First, it tries to resolve the ambiguities left so far by applying knowledge about the context implied in some terms in the query, or about the domain. The domain knowledge, such as State is-a-kind-of Location and Location can-be-preceded-by-preposition IN, are entered by the users in advance and used to help clarify some ambiguities of the input query if necessary.

Secondly, it refines the specification of data items in the query constituents by activating a rule base associated with the user profile. It contains such knowledge as default requested and sorting data items, and default screening conditions for data retrieval based on the facts such as what job the user is working on, what agency the user is in, and the context of the input query.

This task ensures that the system retrieves data that is appropriate for the user, even though his input query may not be well phrased. For example, a user who is working on the projects in a specific fiscal year may pose a query without specifying the fiscal year. By firing the rules encoded in the user profile, the reasoner is able to specify an additional screening condition of a specific year. The retrieved data might otherwise span long periods, up to 15 years or more, which would waste human and computer resources. This rule base is, of course, accessible and may be modified or supplemented by the user, to better tailor the system to the user's perceived needs.

Next, the reasoner checks the blackboard to see if any term used in the input query has meaning associated with procedures or rule bases. If it does, then the reasoner will activate an inference engine which will execute these to generate new facts or data. The blackboard then is updated.

For solutions to complex queries, the reasoner often requires procedures or rule bases. A procedure may expect and require specific types of information, or arguments, such as values of a specific field or fields with specific contexts. The inference engine is able to search for the required information from the input query. If such information is not found in the query or on the blackboard, the system will request the information from the user. Procedures may also be nested for generality and convenience, one calling other procedures or rule bases.

The next task of the reasoner is to determine the processing sequence for generating the answer to the input query. It must determine if database retrieval is required, and if manipulation beyond direct field values retrieval is required, whether the generated data can be generated within the database manager (pre-access processing in that the system must generate instructions to the database manager) or must be imported first and solved within the system (post-access processing). A simple query involves only direct retrieval and reporting, with possible simple pre-access processing for such terms as "difference between" or "sum of". More complex queries, based on procedures, specify computations on data which may or may not be able to be executed in the DBMS's query language. The most complex procedures will direct multiple database retrievals and then look for patterns among retrieved data.

The pre-access processing forms FOCUS program code for calculated data items and flags. It also activates the query generator to generate the FOCUS language using the internal representation of input fields and selection and sorting conditions found in the query and resulted from the execution of procedures and/or rule bases. The communications package then uploads the FOCUS program to the CAPCES database and download the output from it. For retrieval, the system may log onto a mainframe being located in Dallas, Texas, through a dedicated telecommunication network. To save retrieval time, the system can retrieve data from topical databases downloaded previously to the PC. These are downloaded during off hours from the mainframe.

Post-access processing uses data imported from the database to infer additional information. Post-processing can range from a simple arithmetical operation to a pattern search, or a diagnosis or forecast of trends. These are directed with procedures or rule bases.

Chapter 4

Knowledge Representation and Reasoning in the System

Computers have been widely used for number crunching and data processing for the last few decades. Researchers in many areas today try to push use of computers even further by imitating human's intelligence to solve more difficult problems or share more work that is otherwise done by humans. Currently, computers still cannot accept natural language directly as a programming language, since it exists ambiguities both in structure and meaning.

Expert-MCA is designed to derive answers to queries that are expressed in natural language. To derive the answers, Expert-MCA must reason. One strategy for such reasoning process is to establish mechanisms for transforming natural language to an internal representation which facilitates further processing, such as resolving ambiguities in a query, identifying its correct meaning, and processing tasks as requested in the query.

Knowledge representation and reasoning are closely related. As far as knowledge representation and reasoning are concerned, the following issues are essential to the design and implementation of Expert-MCA. How does Expert-MCA translate English sentences, a surface language as used by users to query the system, into an internal representation (language)? How does the internal representation in language process the knowledge that is either literally conveyed by the query, indicated by context in the query, or specified in the profile session to describe the application domain?

The chapter discusses the knowledge representation and reasoning mechanisms provided in Expert-MCA. They serve as a foundation for building the modules that are discussed in the next two chapters.

4.1 Representation Features in Expert-MCA

Representation systems used in Expert-MCA are designed under the following considerations: abstraction in logical representation and physical implementation, flexibility to accommodate changes in views about the real world, facilitation of reasoning process, and enhancement of reasoning capability. In order to facilitate the advantages of abstraction and

flexibility, many layers of representation between users and computers are required. As shown in Fig. 4-1, an object-oriented representation used internally in Expert-MCA serves as a middle layer of representation system between user's sentences and a third generation language Lisp, which might further generate programming code in FOCUS, a fourth generation language.

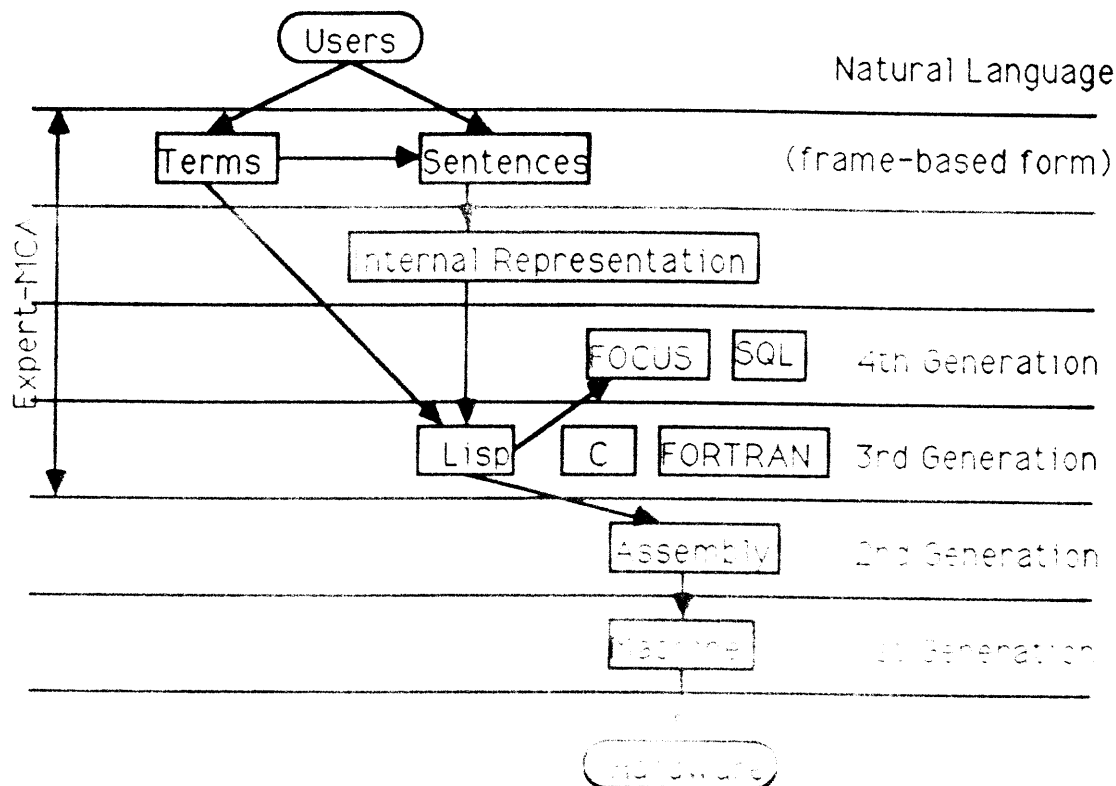


Figure 4-1: Abstraction Layers between Users and Computers in Expert-MCA

The top layer language is used by users. It is a subset of natural language. The basic unit of the top layer language is term, which is an English word or phrase. Query sentences are the language used to ask Expert-MCA for retrieving information. Each query sentence is composed of several terms which are defined by users themselves. The use of a term is given in the next section, while detailed specification in defining a term is given in Appendix B.

The second layer is an internal (or immediate) representation that is constructed with an object-oriented formalism. The internal language needs reasoning mechanisms for processing knowledge that is conveyed in the surface language of the top layer. Underlying

such internal language is a formal computer language, Lisp. In some cases, the formal query language FOCUS is needed because the target database management system CAPCES can only recognize FOCUS for data retrieval.

4.2 Classification of Terms Used in the System

Terms are the basic units that the user uses to communicate with Expert-MCA. A query sentence will be transformed into an internal representation with which Expert-MCA is able to process knowledge for deriving answers. Symbolically, a term is a group of alphanumerical characters which may have spaces between them. Namely, a term can be an English word or phrase, or other alphanumerical symbols which make sense to the user.

A term is usually associated with either an object in the application domain or an operator that is used to relate objects. An object can be an concept, a state, or a physical entity. Each object may be represented or derived by its more refined subobjects in conjunction with some operators. For example, terms such as **cost overrun**, **expected cost**, and **actual cost** are all concepts. However, the concept **cost overrun** can be semantically refined by using the other two concepts. Namely, **cost overrun** can be defined as: **actual cost is greater than expected cost**. Terms can also have multiple meanings. For instance, **MA** is a field value for the field **PRCD** (for program code) in **CAPCES**. It can also be used as a synonym for the state of **Massachusetts**.

The knowledge being extracted from the terms in input sentences is embedded in their definitions stored in the dictionary of the system. In order to help the system understand the meaning of the input sentence and help the user maintain or update the dictionary, terms are classified into six types: **database field name**, **database field value**, **procedure**, **rule base**, **synonym**, and **other word or phrase**. These six term types are abbreviated as **Fieldname**, **Fieldvalue**, **Procedure**, **Rule**, **Synonym**, and **Word/Phrase**, respectively.

A frame-based format is applied to represent a term. More detailed specification for these terms are given in **Appendix B**. The values in slots of term frames can be a single word, a phrase, a simple sentence, a series of production rules, or a series of procedural statements. A term frame has the term being represented as its name. A term type is also specified in a term frame. Term frames have different slot names for different types of terms. Each term frame has two parts: **leading part** and **property part**. The leading part contains the term being represented and its associated term type, while the property part has several elements, each

containing a slot name and its associated values. To save its size, the dictionary does not record the slots with valued unspecified.

Fieldname and Fieldvalue correspond to the two primary data types in CAPCES database. For example, "Massachusetts" is a value for the field STATE; FY is the Fieldname for fiscal year. These two term types can also be viewed as the basic components of a query sentence in meaning. Fieldnames are used to specify different fields of data in the database for printing or sorting data items in reports. On the other hand, Fieldvalues are used to choose specific ranges of data that the user wants to obtain. The term type Synonym consists of terms that have equivalent meanings to other terms. For instance, CA and MA can be defined as Synonyms for the two Fieldvalues California and Massachusetts, respectively.

Word/Phrases are used to represent English words, phrases, or acronyms which have simple meaning in the application domain. By simple meaning here, it indicates that the meaning conveyed in a Word/Phrase is direct, straightforward, not as complicate as Procedures or Rules which have to be processed in order to get its correct meaning. The content of the meaning embodied in a Word/Phrase can be a **contextual phrase**, an arithmetical statement, the term itself, or nothing.

A contextual phrase is a proposition that specifies a condition about a data item or a context. For example, fiscal year is 88, program amount is less than 50000, state code is 24 or 25, actual cost is greater than budgeted cost, and overrun cost is greater than 10000 are contextual phrases. Semantically, a contextual phrase consists of data items, logical operator, and values. Logical operators such as EQ, EQUAL TO, GT, OR, and GREATER THAN stand for terms that specify logical relationships between the terms located prior and next to it.

A data item can be a Fieldname or a derived-item. A **derived-item** is an item whose value is derived according to a statement which consists of one of the followings:

- (1) two CAPCES fields and an arithmetical operator;
- (2) a CAPCES field, another derived-item, and an arithmetical operator;
- (3) tow other derived-items, and an arithmetical operator;
- (4) a CAPCES field, a value, and an arithmetical operator;
- (5) a derived-item, a value, and an arithmetical operator.

For example, the term overrun cost is a derived-item since it is defined as **current working estimate - program amount**, where current working estimate and program amount are two CAPCES fields and "-" is an arithmetical operator. If a Word/Phrase term is defined as a derived-item, it can be used as a **pseudo-field** in Expert-MCA. This is because the way that

a derived-item term performs in Expert-MCA is exactly the same as a CAPCES field. In other words, once a pseudo-field term is defined, mostly with a term type Word/Phrase, the user can conceptually treat it as a Fieldname term and use it to define other terms or pose queries. For example, we may use the pseudo-field (or derived-item) term overrun cost to define another Word/Phrase seriously cost overrun as overrun cost is greater than 1 million.

A Word/Phrase can also be defined as an arithmetical statement. In this case, such Word/Phrase is a derived-item term. An arithmetical statement is composed of several data items that are connected by an arithmetical operator, such as PLUS, MINUS, "+", "/", or "-". For example, overrun ratio can be defined as "current working estimate / program amount."

Since such terms as logical or arithmetical operators are fundamental vocabularies of the internal representation used in the system, they are defined as Word/Phrases with contents identical to the terms themselves. The terms such as GT, LT, EQ, +, and - fall into this kind of Word/Phrases. Some other Word/Phrases may have functional implications in sentences but do not contribute meanings directly for retrieving data from CAPCES. The terms such as prepositional words fall into this group of Word/Phrases. Although nothing enters in their meaning slots (i.e., Definition slot in Word/Phrase term frame), they do provide functional mechanisms to help Expert-MCA interpret input sentences.

Moreover, a term expecting to bind with its neighboring words in the input sentence as a new phrase in meaning can also be defined as a Word/Phrase¹². Within the context of CAPCES database, for example, the contextual phrases "construction completion percent is less than 40" and "design percent is greater than 80" are represented as "des_percent lt C40" and "des_percent gt D80", respectively. The term "construction completion percent" or "design percent" can be defined as a Word/Phrase in which the specification about how to select variable terms is defined in slots of the term frame. Such a specification includes locations of the expected variable terms in a sentence relative to its primary term, their data types, and their value ranges. After such a term as "construction completion percent" is defined, Expert-MCA is able to look for its neighboring words and test on them given the

¹²Such a term is also called split term, which consists of a primary term and some variable terms. The specification for such a term can be referred to in Appendix B.

specification in the term¹³. As such, Expert-MCA can understand the meaning of some values, such as 40 without storing the definitions of them in the dictionary. It also frees the user from frequently converting a constant to a specific type of field values with a special coding format such as converting "40" to "C40" and "80" to "D80" above. The terms "construction completion percent" and "design percent" share the same field "DES_PERCENT," but with different ways of differentiating their values. In this case, the former adds a leading letter "C" to a numerical value, while the latter inserts a leading letter "C."

A Procedure or Rule involves with more complicated meaning and cannot be determined without further processing on the information that is not literally conveyed by each of the terms in the input sentence. Rather its meaning is dynamically determined based on the context in the input sentence, the user's status or working profile, or sequences of data retrieval. A Procedure differs from a Word/Phrase in that the former does not have a predefined properties for expected variable terms. It often derives its value from contextual information available during a query session. The context of terms in a sentence, for example, can be used as an indication for a Procedure to associate with related terms.

A procedure term in a sentence is more like a function which looks for its arguments by context. The arguments may be extracted either from the terms in the same sentence or knowledge sources which contain information about the user and the MCA cycle. A Procedure can be defined as a series of procedural statements which are executed for retrieving, calculating, inferring, using different Rules, or calling other Procedures in the course of answering the user's queries. Procedures can be used to define temporary (or derived) fields, derive new facts, generate customized reports, evaluate work performance, diagnose troubled projects, or forecast project's progress.

A Procedure may be required when a sentence, for instance, contains a term that does not directly match with any existing fields in the CAPCES database. A Procedure can be used to derive the values for the term "design funds" in the sentence "what are the design funds for the projects that are in MA and FY 89", since the data item "design funds" does not exist in the CAPCES database. Rather the values for "design funds" are calculated by a

¹³How this is done will be detailed in Section 5.3. For example, given the input sentence, "Show the projects with construction completion percent less than 40," Expert-MCA will test its following terms, In this case "less than" will be identified as a logical operator "LT" and "40" as a number between 0 and 99. Therefore, variable terms as specified in the definition slot of the term "construction completion percent" will be instantiated.

Procedure based on values of existing data items which are in context related to "design funds."

Rules are knowledge units that support Procedures or other reasoning modules in the system for finding new facts. Each Rule contains a series of production rules. They can be triggered by referring to its name in conjunction with an initial set of working data. A production rule is generally expressed in IF-THEN statements. Rules usually do not appear in input sentences. They are triggered by Procedures or some reasoning mechanisms for various purposes.

4.3 Object-Oriented Representation

In order to obtain requested information, Expert-MCA has to transform input sentences into an internal representation, which is in turn used for knowledge processing. How Expert-MCA allows users to define English words and phrases, or to pose English sentences is essential to its interface performance, while the choice of an internal representation in Expert-MCA has great influence on its reasoning capability internally.

Since we often need domain knowledge to help resolve the ambiguities in an input sentence and identify its correct meaning for further executing tasks, the representation for such knowledge and the reasoning mechanisms under such representation are essential to the capability of the ambiguity resolution and meaning identification. This section and the next two sections address such issues and propose feasible solutions for the knowledge representation and comparable reasoning mechanisms which are suitable to the Military construction domain.

The commonly used relationships between objects include ISA or A-Kind-Of (for classification or specialization), Is-Part-Of or Has-Part (for aggregation), and Is-Member-Of (for grouping). The relationship A-Kind-Of allows us to organize objects in a hierarchical structure. Such hierarchy has been used to great advantage by programs that operate in knowledge domain where the hierarchical nature of the knowledge is important. For objects that can be decomposed into different parts, Is-Part-Of is quite handy in representing aggregation relationships among objects. A-Member-Of is used to represent the relationship between a set and its members.

A robust representation system should be allowed to express a richer set of relationships than the above three, which are always too simple to be able to precisely

specify relationships among objects with rich semantics, such as those in the construction domain. Object relationships should be dynamically created or modified as users need, because system designers cannot predefine all possible relationships that users might need in the future. In other words, we need derivative relationships to specify objects with a dynamic nature.

Basically, derivative relationships are to do with the notion of semantics. Here shows some examples of derivative relationships that we may need. An object can be created by associating multiple objects, as shown previously about the objects **cost overrun project**, **actual cost**, and **budgeted cost**. After deriving an object like **cost overrun project**, one may define another object **troubled project** as a project with **cost overrun** or **time overrun**. Still, one may further define a new object called **seriously troubled project** by looking for projects which meet the conditions that **overrun cost is greater than a critical amount** or **overrun time is longer than a critical time period**.

Classification of object types is fundamental to design of management information system or knowledge-based system, and to operations on such objects in the system [10, 48, 52, 78]. Objects used in Expert-MCA can be classified, in function, as the following types: attributive, procedural, operative, primitive, relational, and derivative.

Attributive objects contain attributive information such as attribute names and attribute values. A special case for attributive objects is that they are primitive concepts, such as Location, State, and Time. Procedural objects contain statements which are used to procedurally operate on knowledge or information that is carried by objects. An access to a procedural object usually is to obtain information for further processing. Operative objects are some predefined objects that are used to operate the processing in the system. These include all mathematical and logical operators. Primitive objects are primitive objects that are created and maintained by the system itself. For example, the objects¹⁴ Transitive-Relation, is-an-instance-of, is-a-kind-of, etc. fall into this type of objects. Users can use primitive objects to define¹⁵ new objects as they see appropriate. These new objects in turn can be used in defining other terms, perhaps together with some primitive terms.

Derivative objects mean that the objects are defined in a way that their meaning or

¹⁴Use of these objects will be exemplified in the next section.

¹⁵This can be defined in the user profile and domain profile in the Profile Session, or terms with the type Rule in the Teaching Session.

attributes are derived from others. They do not have static values for their attributes. Instead, they are dependent on other objects and change dynamically. The use and the function of derivative objects are very similar to external views in a relational database.

Relational objects are used to relate objects. Relational objects in Expert-MCA are an open class, meaning that users create and use relational objects as they see a need. Names of relational objects do not matter their use in reasoning process, as long as they can be used consistently in semantics. For example, the object *Massachusetts* can be related to the objects *state* and *New England* by using the relational objects *is-an-instance-of* and *is-in*, or *Massachusetts is-an-instance-of State* and *Massachusetts is-in New England*, respectively.

The notions like *is-preceded-by*, *is-followed-by*, *is-not-preceded-by*, and *is-not-followed-by* may be used not only as relations, but also as concepts (or objects), which in turn can be associated with other objects by relations [94]. This is why relations used in Expert-MCA are also classified as relational objects. More importantly is that the classification of relational objects allows Expert-MCA to greatly enhance its reasoning capability (Please refer to the next section).

Note that an object can be associated with multiple objects types. For example, *is-concatenated-by* is a relational object in the statement: *<sorting-phrase> is-concatenated-by BY <field>* (Please refer to Section 7.1). It is also a derivational object, meaning that the variable object *<sorting-phrase>* is derived by concatenating the two objects: a string object *BY* and a variable object *<field>*. In implementation, Expert-MCA associates a derivational object with a Lisp function. Therefore the use of an object that can be a relational object and a derivational object is two-fold. It can be used in context driven reasoning to relate objects. It can also be used to derive a value for the object being associated. That how such objects are used is solely depending on the environment in which the objects are activated.

For simplicity, we use **associative object network** to indicate the network which consists of the nodes using the object-oriented representation as described above. Associative object network is similar to semantic networks in concept, but similar to frame-based systems in its representation structure. It allows users to represent objects in conjunction with semantic relationships, which are also a kind of objects. The reasoning capability in the associative object network can be extended well beyond the one commonly used in frame-based representation, because it can be specified by users as they need to customize their problem domain.

4.4 Context Driven Reasoning

The ability to reason (or inference) is one of fundamental criteria for a person or a machine to be intelligent. There are many reasons why we need computer programs or, specifically, knowledge-based expert systems to reason. We are not able to formulate explicitly all the world we can conceive, and encode it to the systems. Nor can we possibly perceive or foresee all the demands that the domain of interest may require. We are also unable to predict all the changes in the environment with which the systems might have to cope [39]. Reasoning is just like an information machine that can extract and use the information implicit in what is explicitly represented. The purpose of the information machine is to create new information, or reach a conclusion, that is otherwise unknown.

From an object-oriented object standpoint, reasoning results should update or instantiate objects with values for their attributes, establish or renew relationships among objects, or both of the two. Conventional programmings have primarily work on computation of values for objects and propagation of such values between objects, while knowledge-based expert systems have more geared to the manipulations on relationships among objects. In other words, the former focuses more on operations with **numerical** values for objects, and the latter emphasizes more on **meaning association** among objects. Assertion of a proposition involved more than two objects is what we mean meaning association here. Usually it takes the form: <object-1> <relational-object> <object-2> or <object-1> <relational-object> <object-2> <object-3>

What most knowledge-based expert systems lack at present is the ability to automatically propagate meaning association among objects. In rule based systems, for example, to infer a proposition in a THEN-part, we have to explicitly specify in the THEN-part the statement in which some objects might be expressed as variables, which will be instantiated only when its counterpart IF-part is true. Without such a predefined, fixed statement for a proposition, the inference engine in the system simply cannot instantiate or infer the proposition for users. In other words, it lacks the ability, or a kind of reasoning knowledge, to automatically propagate meaning association among objects. By automatic propagation in meaning association here, we mean, via an example, that

Given the facts:

Cambridge is-in Great-Boston
Great-Boston is-in New-England

the system should automatically infer that

Cambridge is-in New-England

without specifying the following rule:

```
IF
  <object-1> is-in <object-2>
  <object-2> is-in <object-3>
THEN
  <object-1> is-in <object-3>
```

With this in mind, this research has presented a new mechanism for accomplishing automatic propagation in meaning among objects. Being supported with this mechanism, called context driven reasoning as described in what follows, users can alleviate a lot burden in specifying rules in a knowledge-based expert system.

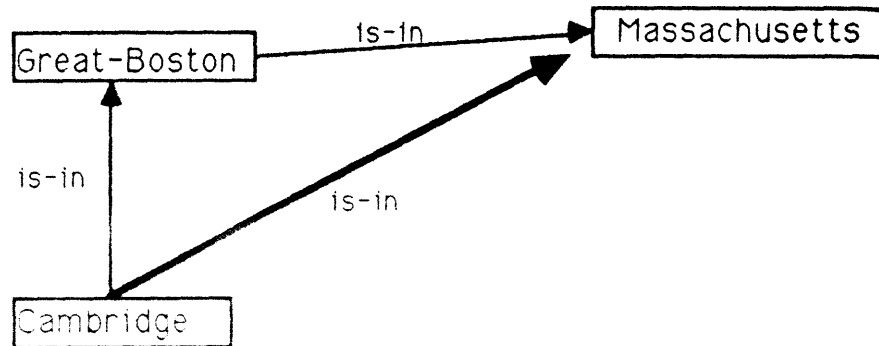
Context driven reasoning is a generalization of the reasoning mechanisms that most frame-based systems have employed for property inheritance. Such reasoning is so named because it makes sense only to those contexts involved or specified in the reasoning mechanism. In other words, the way it reasons is primarily depending on the contexts involved. Reasoning generalization presented here has been inspired by the works of many researchers, such as Maida [57], Winston [94], and Woods [96]. The following sections will discuss three cases of generalization for reasoning, as shown in Figures 4-2, 4-3, and 4-4, respectively. All of these three are used in Expert-MCA.





To illustrate, let us define some relational objects and primitive objects. While users can define any relational object as they see a need to describe the domain of interest, we use the following as exemplary relational objects to illustrate how context driven reasoning works: is-an-instance-of, is-a-kind-of, is-a-member-of, is-part-of, is-a-unit-of, has-property, is-in, and can-be-preceded-by-preposition. Again, the names of such relational objects are not significant in their use, as long as users use them consistently in semantics.

Two primitive objects are described here: **Transitive-Relation** and **transit-over**. The relation specified by an instance object of **Transitive-Relation** is transitive. The concept of **transit-over** indicates that a relationship between one pair of objects can transit over to another pair. As a matter of fact, many logical operators, such as ">", ">=", "<", "<=", and "=", are some instances of the class object **Transitive-Relation**. For instance, the relation specified by the logical operator gt (for greater than, i.e., ">") is transitive, as illustrated as follows:

Case 1:

Meaning propagation along same kind of relational objects



Legend.  Given Relationship
 Deduced Relationship
 Object
 Name of a Relational Object

<Object-X> <relational-object-1> <Object-Y>
<Object-Y> <relational-object-1> <Object-Z>
<Relational-Object-1> is-an-instance-of Transitive-Relation

Then

<Object-X> <relational-object-1> <Object-Z>

Figure 4-2: Simple Reasoning Transitivity

If
 X gt Y
 Y gt Z
Then
 X gt Z

where
 X, Y, and Z are numerical variables,
 gt is the logical operator ">".

In what follows, we will present three kinds of reasoning knowledge which can be used in any domain to automatically propagate meaning association among objects, i.e., simple reasoning transitivity, forward-transit reasoning transitivity, and backward-transit reasoning transitivity.

4.4.1 Simple Reasoning Transitivity

To generalize meaning transitivity over relational objects, three categories of generalized reasoning knowledge can be identified. The first one, called simple reasoning transitivity, is a generalized reasoning knowledge over the same relational objects. In the second one, meaning transitivity occurs via a forward-transit direction over different relational objects, while in the third one it occurs via a backward-transit direction over different relational objects. These two are called forward-transit reasoning transitivity and backward-transit reasoning transitivity, respectively. In Expert-MCA the three kinds of reasoning knowledge is implemented as a system's reasoning knowledge, which will be feed into its inference engine for inference processing.

Some convention for the symbols used for reasoning transitivity is described as follows: anything inside the arrow brackets "<>" is treated as an object variable, which can be replaced by any object; a symbol with lower-case characters is a specific relational object; a symbol with initial letters capitalized is a specific object name. As mentioned in Section 4.3, a small set of primitive objects is solely defined by the system. Users are not supposed to define or modify the primitive objects, which are stored in the domain profile file. Contents of the file can be echoed in the Profile Session of Expert-MCA.

For the purpose of explanation, simple reasoning transitivity can be described in terms of the following rule:

Simple Reasoning Transitivity:

If

<Object-X> <Relational-Object> <Object-Y>
<Object-Y> <Relational-Object> <Object-Z>
<Relational-Object> is-an-instance-of Transitive-Relation
Then
<Object-X> <Relational-Object> <Object-Z>

Where

<Object-X>, <Object-Y>, and <Object-Z> are object variables,
<Relational-Object> is a variable for relational objects,
is-an-instance-of is a specific relational object,
Transitive-Relation is a primitive object.

In Expert-MCA, user can in advance specify in the Profile Session the proposition "Is-In is-an-instance-of Transitive-Relation" as a piece of domain knowledge.

For example, when given the facts:

Cambridge is-in Great-Boston
Great-Boston is-in Massachusetts

the system will fetch available reasoning knowledge, including the generalized knowledge about meaning transitivity, and then infers that Cambridge is-in Massachusetts by applying the simple meaning transitivity rule described above.

4.4.2 Forward-Transit and Backward-Transit Reasoning Transitivity

To further generalize the notion of transitivity between objects via a single relational object, we may use the primitive object transit-over to express the transitivity over multiple relational objects. Let us first examine an example for use of the logical operators gt (for greater than) and eq (for equal to) as follows:

If
X gt Y
Y eq Z
Then
X gt Z

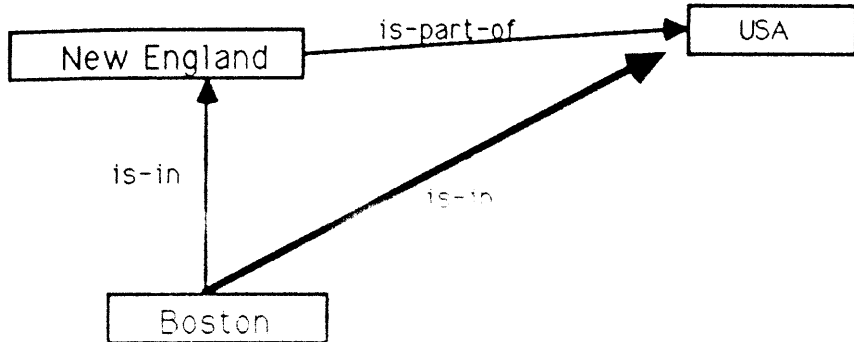
Where

X, Y, and Z are numerical variables,
gt is the logical operator ">",
eq is the logical operator "=".

To generalize the notion of transitivity between objects over multiple relational objects, we can express the ideas about forward-transit reasoning transitivity and backward-transit reasoning transitivity as follows:

Case 2:

Meaning association transits forward over different relational objects



Legend:

- Given Relationship
- Deduced Relationship
- Object
- Name of a Relational Object

If

<Object-X> <relational-object-1> <Object-Y>

<Object-X> <relational-object-2> <Object-Z>

<Relational-Object-1> forward-transit-over <Relational-Object-2>

Then

<Object-X> <relational-object-1> <Object-Z>

Figure 4-3: Forward-Transit Reasoning Transitivity

Forward-Transit Reasoning Transitivity:

If <Object-X> <Relational-Object-1> <Object-Y>
 <Object-Y> <Relational-Object-2> <Object-Z>
 <Relational-Object-1> forward-transit-over <Relational-Object-2>
Then
 <Object-X> <Relational-Object-1> <Object-Z>

Backward-Transit Reasoning Transitivity:

If <Object-X> <Relational-Object-1> <Object-Y>
 <Object-Y> <Relational-Object-2> <Object-Z>
 <Relational-Object-2> backward-transit-over <Relational-Object-1>
Then
 <Object-X> <Relational-Object-2> <Object-Z>

To illustrate the inference mechanisms described above, some relational objects are exemplified as follows.

1. Is-a-kind-of backward-transit-over Is-a-unit-of:

If State is-a-unit-of Country
 Country is-a-kind-of Location
 Is-a-kind-of backward-transit-over Is-a-unit-of
Then
 State is-a-kind-of Location

2. Is-in forward-transit-over Is-part-of:

If Boston is-in New-England
 New-England is-part-of USA
 Is-in forward-transit-over Is-part-of
Then
 Boston is-in USA

3. Is-a-kind-of backward-transit-over Is-an-instance-of:

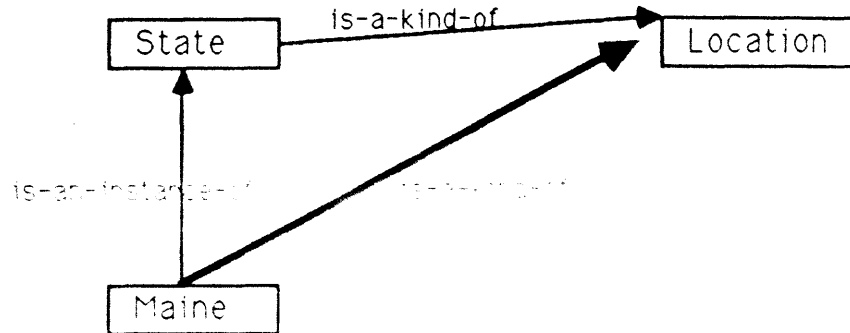
If MA is-an-instance-of State
 State is-a-kind-of Location
 Is-a-kind-of backward-transit-over Is-an-instance-of
Then
 MA is-a-kind-of Location

4. Has-property backward-transit-over Is-member-of:

If PC is-member-of Computer
 Computer has-property CPU
 Has-property backward-transit-over Is-member-of
Then
 PC has-property CPU

Case 3:

Meaning association transits backward over different relational objects



Legend: Given Relationship
 Deduced Relationship
 Object
 Name of a Relational Object

$\langle \text{Object-X} \rangle \langle \text{relational-object-1} \rangle \langle \text{Object-Y} \rangle$
 $\langle \text{Object-X} \rangle \langle \text{relational-object-2} \rangle \langle \text{Object-Z} \rangle$
 $\langle \text{Relational-Object-2} \rangle$ backward-transit-over $\langle \text{Relational-Object-1} \rangle$
Then
 $\langle \text{Object-X} \rangle \langle \text{relational-object-2} \rangle \langle \text{Object-Z} \rangle$

Figure 4-4: Backward-Transit Reasoning Transitivity

5. Has-property backward-transit-over Is-an-instance-of:

If David's PC is-an-instance-of PC

PC has-property CPU

Has-property backward-transit-over Is-an-instance-of

Then

David's PC has-property CPU

6. Has-property-value backward-transit-over Is-an-instance-of:

If David's PC is-an-instance-of PC/AT

PC/AT has-property-value Processor 80286

Has-property-value backward-transit-over Is-an-instance-of

Then

David's PC has-property-value Processor 80286

7. Can-be-preceded-by-preposition backward-transit-over is-a-kind-of:

If MA is-a-kind-of Location

Location can-be-preceded-by-preposition IN

Can-be-preceded-by-preposition backward-transit-over Is-a-kind-of

Then

MA can-be-preceded-by-preposition IN

As you may have found from the above examples, the use of relational objects is involved with semantic notion. In this respect, Expert-MCA does not force a user to conform his or her use on semantic-related objects with the one that is envisioned by its designer. Rather, it leaves a user to decide how he or she will define and use relational objects in specifying propositions for the application domain. Since a proposition in one context is correct but might be wrong in another, user is responsible for verifying the correctness of their propositions within the right context.

Again, the above relational objects and propositions are not necessarily used in Expert-MCA. Most of them serve as illustrated examples for applying the reasoning knowledge about forward-transit and backward-transit reasoning transitivity.

As you will see in Section 6.3.2, examples 3 and 7 above are used to solve an ambiguity about the term MA that the language analyzer may not be able to resolve. Rather its correct meaning is identified by using the context driven reasoning mechanisms.

As such, context driven reasoning can be used to reason any information in any context, as long as relationships between objects, including relational objects and non-relational objects, are specified correctly in the contexts involved. Therefore context driven reasoning can be very powerful, especially in an environment involving complex contexts, such as natural language understanding and managerial domains.

Chapter 5

Language Analyzer

The language analyzer of Expert-MCA is responsible for converting the English query into an internal representation that can be used for further processing to answer the query. This chapter first describes fundamental ideas underlying the design of the language analyzer. These ideas reflect the facts that how the language analyzer is designed, what would be its strength and weakness, and what it should pass to other modules those it cannot do. Next, the modules of the language analyzer are detailed in subsequent sections, together with examples of showing how they work.

5.1 Design of the Language Analyzer

As mentioned in Section 2.2.1, different systems designed for language processing have reflected the facts that their designers view language use differently; that some problems are more important than others and should be resolved first; that different application domains are chosen for best illustrating their key arguments about natural language processing.

As far as language processing for the Expert-MCA project has some unique characteristics that one must take into accounts prior to design of the language analyzer in Expert-MCA. Such characteristics include categories of requested information, sources of the information, patterns of query expression¹⁶. The information requested by most of the users is about: some specific projects that meet some conditions, or some specific data items that might be associated with some conditions. Information for these data items can be retrieved from the CAPCES database fields directly, calculated on the values of these fields, derived from sources other than the database, or processed via a combination of the above.

The expression for such requests is typically arranged in an order beginning with requested data items, followed by a series of conditions for specifying scope of retrieved data instances, then, as an option, followed by a description of how the retrieved data should be

¹⁶Some of exemplary sentences are listed in Section 2.2.2.

sorted. The expression given at fields is always simplified, meaning that it can be incomplete both in structure and meaning. Some of words or phrase may also be complex in meaning, requiring further processing for deriving their values.

The language analyzer in Expert-MCA is not designed with a generalized parser which accepts all kinds of input sentences. To be able to derive answers to user's queries, Expert-MCA must interpret these queries correctly. As long as a parse tree can help interpret the meaning of an input sentence, its format and how it is constructed is not so important. Once a parse tree is constructed, more importantly is how one can use it to help determine correct meaning for the sentence as a whole. To the queries with the patterns as described in the last paragraph, it is much more important that Expert-MCA is able to capture their meaning correctly by utilizing the context and knowledge implicitly indicated in the queries or explicitly supplied by the user and the system.

Therefore, one of major concerns in the development of Expert-MCA, including the language analyzer and reasoner, is how to actually derive answers to user's queries, not so much on how well its parser can create parse trees. Rather, the focus is on how its language analyzer can transform the surface form in English into an internal representation with which the meaning of the surface form can be captured, or facilitate further processing in the reasoner if ambiguities cannot be resolved completely.

The major issues in natural language processing for the user's queries about MCA cycle are how to extract definitions for words and phrases; how to group words into larger phrases lexically; how to form meaning units for such words and phrases syntactically and semantically; and how to resolve ambiguities if necessary.

By knowing that the content of any query is related either to the MCA cycle or data stored in CAPCES database, the language analyzer of Expert-MCA does not need to use all the sentence constituents defined by linguists or to have massive amount of context-free rules to answer the query. Nonetheless, it needs a parsing device that can decompose the sentence into different components and checks various syntactic and semantic constraints within the domain of CAPCES database and FOCUS query language.

The way of using different term types in the language analyzer as constraints is similar to that syntactic constraints are used in the ATN approach [16]. The parsing technique used in Expert-MCA is different from others in that it is designed, among others, to handle a term with multiple meanings, to deal with a term that expects to bind its neighboring terms to form a new phrase, to use function words or contexts for helping identify correct meanings,

The language analyzer of Expert-MCA can be detailed into four modules: **lexical mapping, lexical analysis, syntactic analysis, and semantic analysis**. **Lexical mapping is to find the definition of each word in input sentences by first mapping the words with the terms stored in the dictionary and then extracting the information of the matched terms from the dictionary, while lexical analysis is to form new phrases for split terms¹⁷ or the terms with multiple meanings if possible.**

Next, primarily according to parts of speech associated with the terms, syntactic analysis constructs a parse tree which contains three major query components: requested information of the input sentence, screening conditions for selecting needed data instances, and sorting information for arranging output data. Lastly, semantic analysis is to refine these three into more meaningful ones.

5.2 Lexical Mapping

The dictionary of Expert-MCA uses frames to store the definition of terms. After reading the definition of terms from the dictionary files, Expert-MCA groups the terms into 27 term address directories according to the leading letter of each term. The additional one is for the terms that do not begin with one of the 26 English letters. Each term address directory contains many term address lists, each being for a term. For example, the following list shows part of term address directories for the terms with the leading letters "C" and "M", respectively.

```
*DIR-C*= `(
...
...
((CONGRESSIONAL ACTION FIELD) 27 CAP1)
((CONGRESSIONAL NOTIFICATION SYNONYM) 5 DIC2)
((CONG_APRV_YR FIELD) 28 CAP1)
((CONSTRUCTION WORD/PHRASE) 88 DIC2)
((CONSTRUCTION CATEGORY CODE FIELD) 33 CAP1)
....
....
((CONSTRUCTION COMPLETION DATE FIELD) 61 CAP1)
((CONSTRUCTION COMPLETION PERCENT WORD/PHRASE) 79 DIC2)
```

¹⁷As mentioned in Section 2.2.2, a split term consists of a primary term and its associated variable words in the same sentence, together making a complete set in meaning. Examples for split terms are as follows: fiscal year 87, fiscal year 88, 89 fiscal year, and 90 fiscal year, where fiscal year is the primary term; the numbers 87 88 89 90 are the variable words bound to the primary term.

```
((CONSTRUCTION DIRECTIVE AMOUNT FIELD) 56 CAP1)
....
....
)
*DIR-M* = '(
((M-PDIP FIELD) 29 CAP2)
((M-TEMP_PN FIELD) 43 CAP2)
((MA VALUE) 70 DIC2)
((MA WORD/PHRASE) 66 DIC2)
((MACOM FIELD) 55 CAP1)
((MACOM PRIORITY SYNONYM) 27 DIC2)
((MACOM REMARK FIELD) 40 CAP2)
((MACOM SCOPE FIELD) 41 CAP2)
....
....
)
```

In each of the 27 term address directories, the term address lists are arranged alphabetically¹⁸. Each term address list in turn consists of four parts: the term itself, a term type, an address in the file whose name follows it, and a name of the source file that stores the term. Such organization of term address directories facilitates the mixed use of binary search and linear search for a term in the lexical mapping module.

To illustrate how the language analyzer works in different modules, this chapter uses the following query as an exemplary example:

What are the FY 88 projects in MA and PA with PA over 900 and construction completion percent less than 80?"

Upon receiving the exemplary query, the lexical mapping module starts to look up the dictionary for finding information for each word in the sentence.

It first determines a term address directory for each word, and then tries to map the word with terms within the term address directory. The selected term address directory is the one that has the same leading character as the first character of the word in question. For example, if we are now in looking up the dictionary for the word "construction," the selected term address directory is *DIR-C*, which is shown above.

To retrieve information for each word in the input query, the module will go through the following lexical mapping steps. Whenever the searching for a term is successful, its definition will be pushed into a term list mapping stack.

¹⁸It is according to the characters of the terms only, not the entire term address list.

Steps for Lexical Mapping:

- 1. Binary search for the word:** This is to find a term address list which contains the term identical to the word being looked for through a binary search over a selected term address directory. If it fails, then go to step 3 for looking for phrases. For example, if the word being looked for is "MA", the selected term address directory is *DIR-M*, which is shown previously. Suppose the directory contains 97 terms. The module will try to map the word at the following addresses in the directory: 49, 25, 13, 7, 4. As such, the module finds the term address list: ((MA WORD/PHRASE) 66 DIC2), which means "MA" is a Word/Phrase and it is stored at address 66 in the file DIC2¹⁹.
- 2. Linear search for multiple meanings:** This is to find other term address lists that contain the word in question by a linear search starting at the address where it stopped in step 1. Suppose we continue to search for other terms mapped with the word "MA". It first searches forward for the word beginning at the address ended in the first step. In this case, it finds the list at address 5 ((MACOM FIELD) 55 CAP1)", which is for the term "MACOM", is not the one it needs. Next, it searches for backwards. It finds the list at address 3 ((MA VALUE) 70 DIC2)", which is another term it needs. It then continues to search for another term till it fails. In this case, its search will stop at address 2.
- 3. Linear search for phrases:** This is to find the phrases whose leading word is identical to the word in question and they must appear in the input query. It makes a forward linear searching for phrases beginning at the address where it stopped in step 2 (if step 1 is successful) or in step 1 (if step 1 fails). For example, if we are now in looking up the dictionary for the word "construction," the selected term address directory is *DIR-C*, which is shown previously. Suppose also it has found in step 1 the term address list ((CONSTRUCTION WORD/PHRASE) 88 DIC2)", which is at address 35 of the directory *DIR-C*. As can be seen, it fails in step 2. So in this step, it starts searching at address 36, which is the list ((CONSTRUCTION CATEGORY CODE FIELD) 33 CAP1)". It fails in this case. Then it keeps trying next one till the term address list is beyond a phrase in the query that starts with the word in question. Therefore, it will hit the term address list ((CONSTRUCTION COMPLETION PERCENT WORD/PHRASE) 79 DIC2)", which is for the term construction completion percent. Because the input sentence contains this term, the term address list is then pushed into the term list mapping stack.

If none is matched with the word being looked for, a tag "undefined" will be attached to the word. As such, the definition associated with each word in the input sentence is all pooled into the term list mapping stack. Based on the information in the stack, the module is then able to extract definitions for each term in the input sentence from corresponding dictionary files and addresses in the files. Such information will be stored in a parsing stack.

¹⁹In Expert-MCA's implementation, DIC2 indicates the file ESNDIC2.lsp.

Note that the above steps are used to extract definition for each word in the sentence, even a word is part of a phrase that has been found previously. For example, after going through the above steps for the word "construction", the module has found two terms "construction" and "construction completion percent". The module still goes through the above steps for the word "completion" and "percent".

The module also assigns priorities among definitions for a term which has multiple meanings. The module assigns a higher priority to a definition for a phrase than that for a word/phrase, one phrase with more words than another with less words, one term with term type **Word/Phrase** than another with term type **Value**, a split term²⁰ than a phrase, and one split term expecting more variable words than another expecting less variable words.

The purpose of this priority assignment is two-fold. When there does not have enough information to determine a meaning for a term which has multiple meanings, the one with highest priority will be assumed as its meaning. In the following modules, such as lexical analysis and semantic analysis, a higher priority definition will be processed earlier to resolve the ambiguity for the term. With this arrangement, it might save processing time, since a higher priority definition is more likely a correct one.

The parsing stack now contains a set of definitions for each word in the query. Split terms and the terms with multiple meanings are also collected in the parsing stack if they exist. Each element of the stack is corresponding to the definition for each word in the input sentence. If a term has multiple meanings or leads to phrases, its definition expression will be arranged in an order from a higher priority meaning to lower one.

The parsing stack for the exemplary query given above looks like this²¹:

```
((WHAT WORD/PHRASE) ((PART-OF-SPEECH INTE)))
((ARE WORD/PHRASE) ((PART-OF-SPEECH VERB)))
((THE WORD/PHRASE) ((PART-OF-SPEECH DET)))
((FY WORD/PHRASE)
 ((DEFINITION FY IS ?X) (PART-OF-SPEECH NOUN) (CONTEXT FY)
 (VARIABLE-PART ?X) (VARIABLE-LOCATION NEXT)
 (VARIABLE-TYPE NUMBER) (VARIABLE-RANGE 70 99))
(FY FIELD)
((CAPCES-NAME FY) (CAPCES-ALIAS CFY)
 (ENGLISH-NAME CURRENT FISCAL YEAR) (DEFINITION FY)))
```

²⁰The next section will show an example about how a split term can be formed.

²¹This is part of actual parsing stack created in Expert-MCA for the given query.

((88 VALUE) ((DEFINITION 88) (CONTEXT NUMBER) (PART-OF-SPEECH ADJ)))
((PROJECTS WORD/PHRASE)
((DEFINITION PROJECTS) (PART-OF-SPEECH NOUN) (CONTEXT OBJECT)))
((IN WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((MA WORD/PHRASE)
((DEFINITION STATECD IS 25) (PART-OF-SPEECH NOUN) (CONTEXT STATE))
(MA VALUE)
((IN-WHICH-FIELD PRCD) (DEFINITION PRCD IS MA)))
((AND WORD/PHRASE)
((DEFINITION AND) (PART-OF-SPEECH CONJ)
(CONTEXT LOGICAL-OPERATOR)))
((PA WORD/PHRASE)
((DEFINITION STATECD IS 40) (PART-OF-SPEECH NOUN) (CONTEXT STATECD))
(PA FIELD)
((CAPCES-NAME PROG_AMT) (CAPCES-ALIAS PA)
(ENGLISH-NAME PROGRAM AMOUNT)
(CONTEXT AMOUNT HQDA) (DEFINITION PROG_AMT)))
((WITH WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((PA WORD/PHRASE)
((DEFINITION STATECD IS 40) (PART-OF-SPEECH NOUN) (CONTEXT STATECD))
(PA FIELD)
((CAPCES-NAME PROG_AMT) (CAPCES-ALIAS PA)
(ENGLISH-NAME PROGRAM AMOUNT) (CONTEXT AMOUNT HQDA)
(DEFINITION PROG_AMT)))
((OVER WORD/PHRASE)
((PART-OF-SPEECH CONJ) (DEFINITION GT) (ENGLISH-NAME GREATER THAN)
(CONTEXT LOGICAL-OPERATOR) (SYNONYM-OF GT)))
((900 VALUE)
((DEFINITION 900) (CONTEXT NUMBER) (PART-OF-SPEECH NOUN)))
((AND WORD/PHRASE)
((DEFINITION AND) (PART-OF-SPEECH CONJ)
(CONTEXT LOGICAL-OPERATOR)))
((CONSTRUCTION COMPLETION PERCENT WORD/PHRASE)
((DEFINITION DES_PERCENT ?X "C?Y") (PART-OF-SPEECH NP)
(CONTEXT CONSTRUCTION) (VARIABLE-PART ?X ?Y)
(VARIABLE-LOCATION NEXT NEXT-2)
(VARIABLE-TYPE LOGICAL-OPERATOR NUMBER)
(VARIABLE-RANGE N/A (0 100)))
(CONSTRUCTION WORD/PHRASE)
((DEFINITION CONSTRUCTION) (PART-OF-SPEECH NOUN)))
((COMPLETION UNDEFINED) ((ORIGINAL-STRING "completion")))
((PERCENT WORD/PHRASE)
((DEFINITION 0.01) (PART-OF-SPEECH ADJ) (CONTEXT NUMBER)))
((LESS THAN WORD/PHRASE)
((PART-OF-SPEECH CONJ) (DEFINITION LT) (ENGLISH-NAME LESS THAN)
(CONTEXT LOGICAL-OPERATOR))
(LESS WORD/PHRASE)
((PART-OF-SPEECH CONJ) (DEFINITION LT) (ENGLISH-NAME LESS THAN)

((CONTEXT LOGICAL-OPERATOR) (SYNONYM-OF LT)))
((THAN UNDEFINED) ((ORIGINAL-STRING "than")))
((80 VALUE) ((DEFINITION 80) (CONTEXT NUMBER) (PART-OF-SPEECH ADJ)))

5.3 Lexical Analysis

The purpose of the lexical analysis module is to refine the parsing stack by using lexical information.

It first tries to remove from the parsing stack information redundant words that have been collected in the phrases prior to them. For example, information for such words as "completion", "percent", and "than" are removed, since they appear in the phrases collected previously, i.e., in "construction completion percent" and "less than", respectively. As such, the current parsing stack is updated. Now the parsing stack looks like this:

((WHAT WORD/PHRASE) ((PART-OF-SPEECH INTE)))
((ARE WORD/PHRASE) ((PART-OF-SPEECH VERB)))
((THE WORD/PHRASE) ((PART-OF-SPEECH DET)))
((FY WORD/PHRASE)
((DEFINITION FY IS ?X) (PART-OF-SPEECH NOUN) (CONTEXT FY)
(VARIABLE-PART ?X) (VARIABLE-LOCATION NEXT)
(VARIABLE-TYPE NUMBER) (VARIABLE-RANGE 70 99))
(FY FIELD)
((CAPCES-NAME FY) (CAPCES-ALIAS CFY)
(ENGLISH-NAME CURRENT FISCAL YEAR) (DEFINITION FY)))
((88 VALUE) ((DEFINITION 88) (CONTEXT NUMBER) (PART-OF-SPEECH ADJ)))
((PROJECTS WORD/PHRASE)
((DEFINITION PROJECTS) (PART-OF-SPEECH NOUN) (CONTEXT OBJECT))
((IN WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((MA WORD/PHRASE)
((DEFINITION STATECD IS 25) (PART-OF-SPEECH NOUN) (CONTEXT STATE))
(MA VALUE)
((IN-WHICH-FIELD PRCD) (DEFINITION PRCD IS MA)))
((AND WORD/PHRASE)
((DEFINITION AND) (PART-OF-SPEECH CONJ)
(CONTEXT LOGICAL-OPERATOR)))
((PA WORD/PHRASE)
((DEFINITION STATECD IS 40) (PART-OF-SPEECH NOUN) (CONTEXT STATECD))
(PA FIELD)
((CAPCES-NAME PROG_AMT) (CAPCES-ALIAS PA)
(ENGLISH-NAME PROGRAM AMOUNT) (CONTEXT AMOUNT HQDA)
(DEFINITION PROG_AMT)))
((WITH WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((PA WORD/PHRASE)

((DEFINITION STATECD IS 40) (PART-OF-SPEECH NOUN) (CONTEXT STATECD))
(PA FIELD)
((CAPCES-NAME PROG_AMT) (CAPCES-ALIAS PA)
(ENGLISH-NAME PROGRAM AMOUNT) (CONTEXT AMOUNT HQDA)
(DEFINITION PROG_AMT)))
((OVER WORD/PHRASE)
((PART-OF-SPEECH CONJ) (DEFINITION GT) (ENGLISH-NAME GREATER THAN)
(CONTEXT LOGICAL-OPERATOR) (SYNONYM-OF GT)))
((900 VALUE)
((DEFINITION 900) (CONTEXT NUMBER) (PART-OF-SPEECH NOUN)))
((AND WORD/PHRASE)
((DEFINITION AND) (PART-OF-SPEECH CONJ)
(CONTEXT LOGICAL-OPERATOR)))
((CONSTRUCTION COMPLETION PERCENT WORD/PHRASE)
((DEFINITION DES_PERCENT ?X "C?Y") (PART-OF-SPEECH NP)
(CONTEXT CONSTRUCTION) (VARIABLE-PART ?X ?Y)
(VARIABLE-LOCATION NEXT NEXT-2)
(VARIABLE-TYPE LOGICAL-OPERATOR NUMBER)
(VARIABLE-RANGE N/A (0 100))))
((LESS THAN WORD/PHRASE)
((PART-OF-SPEECH CONJ) (DEFINITION LT) (ENGLISH-NAME LESS THAN)
(CONTEXT LOGICAL-OPERATOR)))
((80 VALUE) ((DEFINITION 80) (CONTEXT NUMBER) (PART-OF-SPEECH ADJ)))

Secondly, it tries to lexically resolve the ambiguities that are inherent in a split term or a term with multiple meanings. To form a split term with its associated meaning, the module tries to find a primary term and then bind it with qualified variable terms, which may be located either after or prior to the primary term. In other words, the module scans the current parsing stack to see if there is any term that has been defined to associate with other variable words. If such a primary term is found, the module collects the conditions that specify where to find these variable words, and what characteristics these words should have. These conditions are stored in slots Variable-Location, Variable-Type, and Variable-Range.

The module checks whether or not the input sentence has contained the words that meet all the conditions as specified. If no such variable word is found, then the primary term with its definition is removed from the parsing stack. If such a variable word is found, the primary term and its associated variable words will be collapsed into a new term, together with an instantiated meaning. At the same time, the information for the previously matched variable words will be removed from the parsing stack.

Given the exemplary query, the module will sequentially work on the two primary terms "FY" and "construction completion percent". The word "FY" is identified as a primary

term, since its Variable-Part slot is not empty. The two words "FY" and "88" will be collapsed into a new phrase "FY 88" with a meaning "FY IS 88" being instantiated by its associated variable word 88, because the word NEXT to "FY" is "88", which is a number and between 70 and 99.

Similarly, the three terms "construction completion percent", "less than", and "80" will be collapsed into a new phrase "construction completion percent less than 80" with a meaning "DES_PERCENT LT C80." This is done as follows. In its Variable-Part slot, the term frame indicates that it expects to bind two variable terms, called ?X and ?Y, respectively. These two variable terms should be located, as indicated in its Variable-Location slot, next and next two to the term. In this case, they are "less than" and "80," respectively. The module continue to check if these two variable terms meet the conditions specified in its Variable-Type and Variable-Range slots. Namely, it checks if "less than" is a logical operator, and if "80" is a number with a value between 0 and 100. When the content in a Variable-Range slot is "N/A" (for not available) or nil (for empty), the module does not check the variable term for its value range. Because the tests for these two variable terms are all successful, the three terms "construction completion percent," "less than," and "80" are then collapsed into a new term with a new definition by instantiating the variables ?X and ?Y in the Definition slot for the primary term "construction completion percent."

Now the parsing stack looks like this:

```
((WHAT WORD/PHRASE) ((PART-OF-SPEECH INTE)))
((ARE WORD/PHRASE) ((PART-OF-SPEECH VERB)))
((THE WORD/PHRASE) ((PART-OF-SPEECH DET)))
((FY 88 WORD/PHRASE)
 ((DEFINITION FY IS 88) (PART-OF-SPEECH NP) (CONTEXT FY)))
((PROJECTS WORD/PHRASE)
 ((DEFINITION PROJECTS) (PART-OF-SPEECH NOUN) (CONTEXT OBJECT) ))
((IN WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((MA WORD/PHRASE)
 ((DEFINITION STATECD IS 25) (PART-OF-SPEECH NOUN) (CONTEXT STATE))
 (MA VALUE)
 ((IN-WHICH-FIELD PRCD) (DEFINITION PRCD IS MA)))
((AND WORD/PHRASE)
 ((DEFINITION AND) (PART-OF-SPEECH CONJ)
 (CONTEXT LOGICAL-OPERATOR)))
((PA WORD/PHRASE)
 ((DEFINITION STATECD IS 40) (PART-OF-SPEECH NOUN) (CONTEXT STATECD))
 (PA FIELD)
 ((CAPCES-NAME PROG_AMT) (CAPCES-ALIAS PA)
 (ENGLISH-NAME PROGRAM AMOUNT))
```

((CONTEXT AMOUNT HQDA) (DEFINITION PROG_AMT)))
((WITH WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((PA WORD/PHRASE)
((DEFINITION STATECD IS 40) (PART-OF-SPEECH NOUN) (CONTEXT STATECD))
(PA FIELD)
((CAPCES-NAME PROG_AMT) (CAPCES-ALIAS PA)
(ENGLISH-NAME PROGRAM AMOUNT)
(CONTEXT AMOUNT HQDA) (DEFINITION PROG_AMT)))
((OVER WORD/PHRASE)
((PART-OF-SPEECH CONJ) (DEFINITION GT) (ENGLISH-NAME GREATER THAN)
(CONTEXT LOGICAL-OPERATOR) (SYNONYM-OF GT)))
((900 VALUE)
((DEFINITION 900) (CONTEXT NUMBER) (PART-OF-SPEECH NOUN)))
((AND WORD/PHRASE)
((DEFINITION AND) (PART-OF-SPEECH CONJ)
(CONTEXT LOGICAL-OPERATOR)))
((CONSTRUCTION COMPLETION PERCENT LESS THAN 80 WORD/PHRASE)
((DEFINITION DES_PERCENT LT "C80")
(PART-OF-SPEECH NP) (CONTEXT CONSTRUCTION)))

To resolve unrecognized words in an input sentence, three methods can be applied. The first one is to find their relative locations in a sentence to other recognized terms, and then bind them with the recognized terms. Checking their data types and trying to bind with other recognized terms is the second method. By making use of these two methods, the user does not need to define all the words used in the system. As a result, the saving in dictionary space and in time of entering the definition for such words may be a great payoff. The last method is to echo the unrecognized words on screen for the user to edit if they are typos, or to define if they are new terms.

As a matter of fact, construct of split terms makes use of the first two methods. Split terms can be used to resolve unrecognized terms. For example, if the word "Woodland" is not defined in Expert-MCA, the module is able to resolve this word by forming a new phrase "FT Woodland" with a meaning "STATION IS FORT WOODLAND," given a query consisting of the words "... FT Woodland ..." and a definition for the term "FT" as

((FT WORD/PHRASE)
((DEFINITION STATION IS "FORT ?X") (PART-OF-SPEECH NOUN)
(CONTEXT STATION) (VARIABLE-PART ?X) (VARIABLE-LOCATION NEXT)
(VARIABLE-TYPE STRING) (VARIABLE-RANGE N/A))))

If there are unrecognized terms left at this point, Expert-MCA has to ask the user to edit or define them. This will result in a rerun session through the lexical mapping and lexical analysis modules again. Otherwise the lexical analysis module passes the current parsing stack to the syntactic analysis module.

5.4 Syntactic Analysis

The syntactic analysis module is responsible for decomposing a query into several components by using information mainly about part of speech for each term in the query. The purpose of user's queries is to retrieve data from the CAPCES database, to derive or analyze retrieved data, or to request information related to MCA cycle or the CAPCES database. Therefore, the user first has to specify in his or her queries the data items needed. However, he or she often does not have to retrieve all the possible data instances of requested data items. Rather what the user needs is only a subset of the data instances for the requested data items. Thus, to specify screening conditions for selecting an appropriate subset of data instances is the second component in user's queries. For ease of data reading or comparison, an output report is often arranged in a specific order to fit user's needs. As a result, any query made by the user always conveys the information consisting of the following three query components: requested data items (or printing items), screening conditions (or selecting conditions), and sorting data items.

In reality, queries made by users who routinely work on some tasks at fields are often incomplete syntactically and semantically. A generalized parser, mostly working from left to right sequentially, will reject such ungrammatical queries. In order to better capture information available from given input queries, the language analyzer is designed to handle sentences that may be incomplete or loosely follow English grammar, i.e., near natural language queries. It assumes that most of input queries follow the pattern: requested data items, screening conditions, and sorting data items. As a trade-off, the language analyzer is not designed to parse all kinds of English sentences, nor is it able to deal with all complex embedded sentences²².

Once a query has been parsed by the language analyzer and the reasoner, Expert-MCA provides a chance²³ for the user to correct its understanding about the query. By doing so, the system still continues to work on ungrammatical sentences without giving up and asking the user to retype a new query, since the user still has a chance to modify his or her queries or correct Expert-MCA's misunderstanding.

²²An embedded sentence is a sentence used to modify a noun or phrase (or other part of speech) in a primary sentence. Usually it starts at or follows a relative pronoun. For example, the sentence "which is designed by District Engineers" is an embedded sentence used to modify the word "housing" in the sentence "Show me the projects that are in MA and for housing which is designed by District Engineers."

²³This is done in the interface task Understanding-Correction. Please see Chapter 8.

With the terms in the query are all defined, the syntactic analysis module processes to separate the parsing stack into three query components. This is done by following a set of heuristic rules²⁴. The major part of the rules²⁵ can be simplified as follows:

Rules for Syntactic Analysis:

1. This rule is to find terms for requested data items group. At beginning, the searching mode is set to "in-requested-item-part." The terms prior to the first noun phrase or noun (inclusive) are identified as the requested data items group. For example, given the query as in the previous sections: "What are the FY 88 projects in MA and PA with PA over 900 and construction completion percent less than 80?" the module searches for a noun (not including pronoun) or noun phrase starting from the first term, in this case "What". It fails. So it goes to the next, which is "are". It keeps going till hitting the one "FY 88", which is a noun phrase. Therefore, the four terms "What" "are" "the" "FY 88" are pushed into a requested-item stack. Note that the requested-item stack, the screening-condition-stack and the sorting-item stack (the last two will be discussed in subsequent rules) are all composed of the information list for the terms included in any of the three stacks. Then the module goes to the next rule.
2. This rule is to find another requested data item if possible. If the first noun phrase or noun is followed by another noun phrase or noun, or followed by a conjunctive "and" and another noun phrase or noun, then the requested-item stack is replaced with the new one which starts from the first term to the last term checked so far. For the exemplary example, this rule will find that "projects" is a noun. Therefore the new requested-item stack is replaced with these terms: "What" "are" "the" "FY 88" "projects". This rule will continue to work on the next term of the input query until it fails. It goes to the next rule once it fails.
3. This rule is to find an indicator of starting screening conditions. If the term next to the requested data items group collected so far is a relative pronoun such as "that" or "which", then the terms prior to this relative pronoun is identified as the requested data items and the current mode will be set to "in-screening-part". After this check, it goes to the next rule.
4. This rule is also to find an indicator of starting screening conditions. If the term next to the requested data items group collected so far is a preposition, then the terms prior to this preposition is identified as the requested data items group and the current mode will be set to "in-screening-part". For the given exemplary query, this rule finds that "in" is a preposition. Therefore, the searching mode for this query is set to "in-screening-part". After this check, it goes to the next rule.

²⁴These are not production rules in a rule based system.

²⁵This module is also implemented in Lisp procedures.

5. This rule is to check an indicator of starting sorting data items. If one of the following keywords is detected in the remaining sentence: "sorted by", "ordered by", "for each", or "by", then the terms next to this keyword are identified as the sorting data items group and the ones prior to it are identified as either the screening conditions group (if the current mode is "in-screening-part") or the requested data items group (if the current mode is "in-requested-item-part"). If it fails to find any of these keywords, it sets the remaining sentence as the screening conditions group (if the current mode is "in-screening-part") or the requested data items group (if the current mode is "in-requested-item-part"). In the exemplary example, no such a keyword is found, so the remaining sentence "in MA and PA with PA over 900 and construction completion percent less than 80" is identified as the screening conditions group, and the information list for these terms is pushed into the screening-condition stack. The sorting-item stack is set to empty.

Now information collected in the three stacks looks like this:

Requested-Item Stack:

((WHAT WORD/PHRASE) ((PART-OF-SPEECH INTE)))
((ARE WORD/PHRASE) ((PART-OF-SPEECH VERB)))
((THE WORD/PHRASE) ((PART-OF-SPEECH DET)))
((FY 88 WORD/PHRASE)
((DEFINITION FY IS 88) (PART-OF-SPEECH NP) (CONTEXT FY)))
((PROJECTS WORD/PHRASE)
((DEFINITION PROJECTS) (PART-OF-SPEECH NOUN) (CONTEXT OBJECT)))

Screening-Condition Stack:

((IN WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((MA WORD/PHRASE)
((DEFINITION STATECD IS 25) (PART-OF-SPEECH NOUN) (CONTEXT STATE))
(MA VALUE)
((IN-WHICH-FIELD PRCD) (DEFINITION PRCD IS MA)))
((AND WORD/PHRASE)
((DEFINITION AND) (PART-OF-SPEECH CONJ)
(CONTEXT LOGICAL-OPERATOR)))
((PA WORD/PHRASE)
((DEFINITION STATECD IS 40) (PART-OF-SPEECH NOUN)
(CONTEXT STATECD))
(PA FIELD)
((CAPCES-NAME PROG_AMT) (CAPCES-ALIAS PA)
(ENGLISH-NAME PROGRAM AMOUNT)
(CONTEXT AMOUNT HQDA) (DEFINITION PROG_AMT)))
((WITH WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((PA WORD/PHRASE)
((DEFINITION STATECD IS 40) (PART-OF-SPEECH NOUN)
(CONTEXT STATECD))
(PA FIELD)
((CAPCES-NAME PROG_AMT) (CAPCES-ALIAS PA)

(ENGLISH-NAME PROGRAM AMOUNT)
(CONTEXT AMOUNT HQDA) (DEFINITION PROG_AMT)))
((OVER WORD/PHRASE)
((PART-OF-SPEECH CONJ) (DEFINITION GT)
(ENGLISH-NAME GREATER THAN)
(CONTEXT LOGICAL-OPERATOR) (SYNONYM-OF GT)))
((900 VALUE)
((DEFINITION 900) (CONTEXT NUMBER) (PART-OF-SPEECH NOUN)))
((AND WORD/PHRASE)
((DEFINITION AND) (PART-OF-SPEECH CONJ)
(CONTEXT LOGICAL-OPERATOR)))
((CONSTRUCTION COMPLETION PERCENT LESS THAN 80 WORD/PHRASE)
((DEFINITION DES_PERCENT LT "C80")
(PART-OF-SPEECH NP) (CONTEXT CONSTRUCTION)))

Sorting-Item Stack:

(nil)

The noun phrase or noun indicated in the above five rules can be a Fieldname, a Word/Phrase, a Synonym, or a Procedure, plus its modifiers if any. The modifiers, which can be Fieldvalues, Synonyms, and Word/Phrase, are usually located in front of nouns which may be connected by the conjunctive AND. The results after running through above rules may require additional refinement in the semantic analysis module. For instance, based on Rules 1 and 2 above, the terms "what are the FY 88 projects" are identified as the requested data items. However, this identification is not absolutely correct, because "FY 88" is not a requested data item. Rather, it stands for a screening condition²⁶ "FY IS 88". As can be imagined, the results after running through the above rules for some cases may not be appropriate. These problems will be discussed and resolved in the semantic module.

After the entire query sentence is successfully decomposed into three query components (i.e., the parsing stack is decomposed into the three stacks: requested-item stack, screening-condition stack, and sorting-item stack.), they are then transferred to the semantic analyzer.

The information used in the module is solely coming from the terms literally (including parts of speech and the terms themselves). However, many presumptions and background contexts exist in a communication process from time to time. Such kind of

²⁶How to identify a screening condition in a query is described in the next section.

presumptions or contexts cannot be obtained from individual terms alone. Therefore, further processing is required. The ambiguity that cannot be resolved in this module will be left to the semantic analysis module, which utilizes non-literal information such as term contexts in sentences, jobs or responsibilities of the user, and the domain knowledge of the MCA cycle as a whole.

5.5 Semantic Analysis

The semantic analyzer is responsible for semantically refining each of the query components identified in the syntactic analysis module. It scans the meaning of each term and uses the characteristics of the six term types²⁷ defined in Expert-MCA to refine the query components. One key idea used in this module is how one can identify or form a contextual phrase²⁸.

The algorithms used to analyze the requested data items (or printing items) group and sorting data items group are relatively straightforward. On the contrary, the one for refining the screening conditions into a series of contextual phrases is more complex.

First, the module removes dummy terms from the three stacks. The terms that do not contribute to the process of data retrieval from CAPCES database will be removed from the three components (i.e., the three stacks). Such terms do not contain any information in their Definition slots²⁹. One exception to this is prepositions, which are function words providing functional mechanisms for helping identify meaning for the phrases following them. Such terms, for example, include THE, ARE, WHAT, SHOW, DISPLAY, ME, etc.

Secondly, the module tries to capture the action about how to report the data items requested. When printing data items in reports, there have many action options about how to handle the requested data items. The query language FOCUS provides such options as print, list, count, and sum (Please refer to Section 7.1.1.). The default action used in Expert-MCA is PRINT. However, the terms such as "How much" and "how large" can be defined as "sum amount," meaning that the query starting with one of these terms may want to sum a data

²⁷Please refer to Section 4.2.

²⁸Please also refer to Section 4.2.

²⁹Information for each term is stored in a frame with several slots. Please see Section 4.2.

item (or data items) with a context "Amount." On other cases, "how many" can be defined as "Count." When one of such key terms is detected or not, a corresponding action will be recorded³⁰. At the same time, the term indicating how to print requested data items will be removed from the requested-item stack.

Given the exemplary query as described above, the modified requested-item stack now contains information looks like this:

((FY 88 WORD/PHRASE)
((DEFINITION FY IS 88) (PART-OF-SPEECH NP) (CONTEXT FY)))
((PROJECTS WORD/PHRASE)
((DEFINITION PROJECTS) (PART-OF-SPEECH NOUN) (CONTEXT OBJECT)))

Thirdly, the module may move some terms in the modified-requested-item stack to the screening-condition stack, if these terms indicate that they are screening conditions. The data items requested by the user usually consist of Fieldnames or nouns that have the term type Word/Phrase and their modifiers. Most of the modifiers are adjectives (such as average, highest, latest, and lowest), nouns, or noun phrases. If a modifier is a noun or noun phrase whose Definition slot contains a contextual phrase, logically the modifier should be another screening condition for the data items requested. Therefore such a modifier will be moved to screening-condition stack. For example, currently the modified-requested-item stack has two terms "FY 88" "projects," where "FY 88" is the modifier of "projects" and its Definition slot contains a contextual phrase "FY IS 88." Therefore, the new modified-requested-item stack contains information only for the term "projects."

On the other hand, if a modifier is an adjective, it is very likely used as an arithmetical modifier to restrict how its following data item (or items) should be handled. This is important in constructing phrases in query language. Although Expert-MCA is currently not designed to handle this case, its implementation is straightforward³¹.

Next, the module creates a modified-sorting-item stack after working on the sorting-item stack. The dummy terms such as "sorted by," "by," or "for each" in the sorting-item stack will be removed. What is left in the stack usually contains Fieldnames or temporary fields.

³⁰It is recorded into slot Action-in-Query, a slot in the query frame which is created for recording information derived in the language analyzer.

³¹For example, at this point, the adjective and its following data item should be formed as an adjective phrase with an assertion into a new slot indicating what is the operation according to the adjective. At the time the query language generator constructs FOCUS code, such an indication will be used to form a code, mostly adding a prefix to a field name.

Lastly, the module tries to refine the screening-condition stack around the idea of contextual phrases. No matter how a query expresses its screening conditions in English, each of these conditions semantically should be a contextual phrase, a proposition specifying a range of data instances for a data item (a database field or a temporary field) related to the target database. Therefore the issue is how one can identify screening conditions given information for each term in the screening-item stack.

One strategy for such an identification is to look for patterns of definitions for the terms, but not patterns of the terms themselves. As long as its underlying context can be used to recover its meaning, English expression can be very succinct and flexible so that its patterns can be as diversified as possible. On the other hand, patterns of its underlying meaning for objects tend to be much more simpler. This is because the way we express a simple proposition is limited, especially in expressing a range of a data item within a context of database query.

For example, the following rules show how a contextual phrase can be formed by using information about term types, contexts, and the terms themselves.

1. <Fieldname> <Operator-1> <Fieldvalue> --> <Contextual-Phrase-1>
2. <Contextual-Phrase-1> OR <Fieldvalue> --> <Contextual-Phrase-1>
3. <Fieldname> <Operator-2> <Fieldvalue> --> <Contextual-Phrase-2>
4. <Fieldname> <Operator-3> <Fieldvalue> TO <Fieldvalue> --> <Contextual-Phrase-3>
5. <Contextual-Phrase-1> --> <Contextual-Phrases>
6. <Contextual-Phrase-2> --> <Contextual-Phrases>
7. <Contextual-Phrase-3> --> <Contextual-Phrases>
8. <Contextual-Phrases> <Contextual-Phrases> --> <Contextual-Phrases>

where,

<Operator-1> is IS, EQ, or TO;

<Operator-2> is one of the six words: GT, GE, LT, LE, NE, and CONTAINS;

<Operator-3> is FROM;

<Fieldname> is a field name;

<Fieldvalue> is a value for a field.

Note that the module does not try to map English terms as tokens in using the above rules. Rather, it looks at the patterns of meaning that is stored in the Definition slot for each

term. Let us consider the following simple case. "TX" in the query "Show me the projects in TX for FY 90" will be identified as a contextual phrase, given the term TX being defined as:

```
((TX WORD/PHRASE)
((DEFINITION STATE IS TEXAS) (PART-OF-SPEECH NOUN)))
```

since its meaning expression "STATE IS TEXAS" matches with a contextual phrase pattern (i.e., rule 1 as shown above), where "STATE" is a Fieldname; IS is an Operator-1; and "TEXAS" is a Fieldvalue.

To identify contextual phrases, the semantic analysis module first searches³², from left to right, for an logical operator (such as IS, EQ, LT, LE, etc.) in the Definition slot for each term being collected in the screening-condition stack. One occurrence of a logical operator indicates a specific screening condition. If an operator is detected, the module will compare meaning patterns of its neighboring terms with the rules shown above. If a rule pattern above is matched, corresponding terms will be collapsed into a new phrase with new meaning. This new phrase is also marked with a flag contextual-phrase-true. This process continues until no more operator can be found.

Next, the module is trying to test if any single term which has not been marked with the flag contextual-phrase-true can be identified as a contextual phrase, as shown in the case "TX" above. If it succeeds, information for the term is marked with the flag contextual-phrase-true.

Note that undecidable situations in the above two steps may occur when a term is associated with multiple meanings. To resolve this problem, three ways are feasible. The first one tries to select a meaning which can be successfully combined with its neighboring terms to form a contextual phrase. The second one is using the characteristics of function words such as "AND", "OR", and prepositions. The last one is left to the reasoner, which will tries to resolve the problems by using non-literal information.

After the module has moved "FY 88" to the screening-condition stack for the the exemplary example, the new stack looks like this:

```
((IN WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((MA WORD/PHRASE)
((DEFINITION STATECD IS 25)(PART-OF-SPEECH NOUN)(CONTEXT STATE))
(MA VALUE)
```

³²This module in Expert-MCA for identifying/forming contextual phrases is also implemented in Lisp procedures.

((IN-WHICH-FIELD PRCD) (DEFINITION PRCD IS MA)))
((AND WORD/PHRASE)
((DEFINITION AND) (PART-OF-SPEECH CONJ)
(CONTEXT LOGICAL-OPERATOR)))
(PA WORD/PHRASE)
((DEFINITION STATECD IS 40) (PART-OF-SPEECH NOUN)
(CONTEXT STATECD))
(PA FIELD)
((CAPCES-NAME PROG_AMT) (CAPCES-ALIAS PA)
(ENGLISH-NAME PROGRAM AMOUNT)
(CONTEXT AMOUNT HQDA) (DEFINITION PROG_AMT)))
((WITH WORD/PHRASE) ((PART-OF-SPEECH PREP)))
(PA WORD/PHRASE)
((DEFINITION STATECD IS 40) (PART-OF-SPEECH NOUN)
(CONTEXT STATECD))
(PA FIELD)
((CAPCES-NAME PROG_AMT) (CAPCES-ALIAS PA)
(ENGLISH-NAME PROGRAM AMOUNT)
(CONTEXT AMOUNT HQDA) (DEFINITION PROG_AMT)))
((OVER WORD/PHRASE)
((PART-OF-SPEECH CONJ) (DEFINITION GT)
(ENGLISH-NAME GREATER THAN)
(CONTEXT LOGICAL-OPERATOR) (SYNONYM-OF GT)))
(900 VALUE)
((DEFINITION 900) (CONTEXT NUMBER) (PART-OF-SPEECH NOUN)))
((AND WORD/PHRASE)
((DEFINITION AND) (PART-OF-SPEECH CONJ)
(CONTEXT LOGICAL-OPERATOR)))
((CONSTRUCTION COMPLETION PERCENT LESS THAN 80 WORD/PHRASE)
((DEFINITION DES_PERCENT LT "C80")
(PART-OF-SPEECH NP) (CONTEXT CONSTRUCTION)))
(FY 88 WORD/PHRASE)
((DEFINITION FY IS 88) (PART-OF-SPEECH NP) (CONTEXT FY)))

Suppose the module is looking for a logical operator on this stack. It will hit the one "OVER" whose Definition slot contains a logical operator "GT". The module tries to match rule 3, which needs a fieldname and a fieldvalue prior to and next to "OVER," respectively. Although the term prior to "OVER" is "PA" which has two meanings, the module is looking for a meaning with a term type Field and check if the one after is a Value. It succeeds in this case, so the new phrase will be formed as

((PA OVER 900 WORD/PHRASE)
((DEFINITION PROG_AMT GT 900) (PART-OF-SPEECH NP)
(CONTEXTUAL-PHRASE-P TURE)))

Since no other term is indicated as a logical operator, the module continues to test if

any single term can be identified as a contextual phrase. As a result, the terms "FY 88" and "CONSTRUCTION COMPLETION PERCENT LESS THAN 80" are marked with the flag "contextual-phrase-ture." Next, the module looks for conjunctive such as "AND" or "OR." The term "OR" indicates that its neighboring terms are two contextual phrases about the same context or field, while the term "AND" indicates that its neighboring terms can be two contextual phrases about the same, or different, context or field. In the given stack, there is a conjunctive "AND," with a preceding term "MA" and a following one "PA." Both of "MA" and "PA" have two meanings. Since both are about the field "STATECD," an appropriate meaning for each of these two should be to do with the field "STATECD." Therefore, they are collapsed into a new phrase which looks like:

```
((MA AND PA WORD/PHRASE)
((DEFINITION STATECD IS 25 OR 40) (PART-OF-SPEECH NP)
(CONTEXTUAL-PHRASE-P TURE)))
```

The meaning for the new phrase is coerced from its meaning components: "STATECD IS 25." "AND," and "STATECD IS 40." Note that "AND" is replaced with "OR" in the new definition. The second "AND" is also found. But the module does nothing because its neighboring terms have been marked with the flag "contextual-phrase-ture" already.

In summary, the modified screening-condition stack now looks like this:

```
((IN WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((MA AND PA WORD/PHRASE)
((DEFINITION STATECD IS 25 OR 40) (PART-OF-SPEECH NP)
(CONTEXTUAL-PHRASE-P TURE)))
((WITH WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((PA OVER 900 WORD/PHRASE)
((DEFINITION PROG_AMT GT 900) (PART-OF-SPEECH NP)
(CONTEXTUAL-PHRASE-P TURE)))
((AND WORD/PHRASE)
((DEFINITION AND) (PART-OF-SPEECH CONJ)
(CONTEXT LOGICAL-OPERATOR)))
((CONSTRUCTION COMPLETION PERCENT LESS THAN 80 WORD/PHRASE)
((DEFINITION DES_PERCENT LT "C80")( PART-OF-SPEECH NP)
(CONTEXTUAL-PHRASE-P TURE)))
((FY 88 WORD/PHRASE)
((DEFINITION FY IS 88) (PART-OF-SPEECH NP) (CONTEXT FY)
(CONTEXTUAL-PHRASE-P TURE)))
```

The algorithms described above may fail to resolve some ambiguities. For example, meaning for the term "MA" in the following stack cannot be decided by using the above

semantic analysis, because its two meanings, "STATECD IS 25" and "PRCD IS MA,"³³ are equally qualified as contextual phrases.

((IN WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((MA WORD/PHRASE)
((DEFINITION STATECD IS 25) (PART-OF-SPEECH NOUN)
(CONTEXT STATE))
(MA VALUE)
((IN-WHICH-FIELD PRCD) (DEFINITION PRCD IS MA)))
((WITH WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((PA WORD/PHRASE)
((DEFINITION STATECD IS 40) (PART-OF-SPEECH NOUN)
(CONTEXT STATECD))
(PA FIELD)
((CAPCES-NAME PROG_AMT) (CAPCES-ALIAS PA)
(ENGLISH-NAME PROGRAM AMOUNT)
(CONTEXT AMOUNT HQDA) (DEFINITION PROG_AMT)))
((OVER WORD/PHRASE)
((PART-OF-SPEECH CONJ) (DEFINITION GT)
(ENGLISH-NAME GREATER THAN)
(CONTEXT LOGICAL-OPERATOR) (SYNONYM-OF GT)))
(900 VALUE)
((DEFINITION 900) (CONTEXT NUMBER) (PART-OF-SPEECH NOUN)))

Such kind of ambiguities and some others unsolved must leave to the reasoner. For example, if no other literal information can be used to help identify a correct meaning, then context driven reasoning must be used in the reasoner to find an appropriate context that in turn helps resolve the ambiguity of multiple meanings of a term, as the term MA in the sentence "Show me the projects in MA" illustrated in Section 6.2

At this point, the query is decomposed into three refined query components. The internal representation collected in the three stacks will then posted onto an information blackboard. The reasoner then utilizes different knowledge sources, such as the knowledge about databases, the user, and the MCA domain, to answer the query.

³³It means state code is 25 and program code is MA (for Military Army projects).

Chapter 6

Knowledge Reasoner

A blackboard structure used in Expert-MCA is introduced in the section, followed by a description of how the reasoner works step by step; including use of contextual information to solve query ambiguities, use of user and domain profile information to refine query components, execution of rule based inference and procedure processing for complex tasks.

6.1 Structure of the Blackboard in Expert-MCA

The blackboard concept is also employed in Expert-MCA. The information stored on the blackboard can be classified into two categories: static and dynamic. Static information means the information does not change at all once it is created, mostly at the time of login. Such information includes user's profile information, domain related knowledge and information. Dynamic information is the information being created at the time when a query is made, and probably on the way of deriving the answer to the query. Such information includes user's input queries, user's corrections to the Expert-MCA's understanding of the queries, and a modification of generated FOCUS code. Dynamic information may change

Most of time in the processing of a user's input sentence, the blackboard serves as a working area for information updating/exchange and its information is commonly shared by different modules of Expert-MCA. The various knowledge sources attached to the blackboard includes the modules of lexical mapping, syntactic analysis, semantic analysis, the reasoner, the query language generator, user profile, and practical knowledge used in the MCA cycle. The control mechanism of the blackboard is preset by the designer in Expert-MCA³⁴ Its primary functions are to assign various modules appropriate tasks, such as how to understand a user's query, how to supplement additional or default information to the query in order to make a proper data retrieval, how to select a target database locally in the PC or to communicate with the mainframe, and how to arrange the sequence of producing

³⁴Expert-MCA does not fully implement and use all the concepts that a blackboard structure can provide, since the knowledge sources surrounding the reasoner are not so many and their roles are quite distinct.

final answers to the user. The data structure of the blackboard is like a frame with many slots storing various information and instantiated by some global variables.

As mentioned already, user's profile information is first sent to the blackboard at the start of entering Expert-MCA. It contains user's backgrounds in terms of levels of sophistication about computer applications in general and the MCA processing knowledge in particular. It can also hold user's status in terms of his or her roles and responsibilities, By checking the information on the blackboard, Expert-MCA often directs appropriate steps in solving problems. For example, if a user is not familiar with FOCUS query language, Expert-MCA will not provide the user with a screen in which correction or modification of FOCUS code can be made.

On the other hand, the blackboard also contains the information about how to make supplemental adjustment to the understanding of user's queries.. This is conducted first by checking user's status (or roles and responsibilities) and the understanding of a query derived by the language analyzer, then by adding additional information to the queries. For example, Expert-MCA may add a condition of specific years for selected data instances. It can also find appropriate contexts for solving ambiguities of the queries.

6.2 Construct of the Reasoner

The primary knowledge sources used to help identify meaning for a query in Expert-MCA include information about the CAPCES database, user profile, and the heuristic knowledge used in the MCA cycle. The problem is how such knowledge sources can be used cooperatively and in what order or sequence.

In general, a query would specify the nature of the results desired, that is, the printing data items, sorting data items, and screening conditions. This information, if not present in the query, might be either preset by the user or is inferred by the system. In conjunction with the facts embedded in the input query, the system infers more information based on context of the input query, and knowledge about the user and the MCA application domain. This more intelligent system would require less from the user and would therefore be easier to use.

As discussed in Section 2.4.3, use of a KBES and DBMS may include the following three strategies: enhancements of existing systems, coupling of independent systems, and technology integration [81]. In the first strategy, providing CAPCES database with an input

system with reasoning capabilities can only solve problems prior to data retrieval. Knowledge for interpreting output and then generating more sophisticated and useful information to the user in the MCA could not be included. Thus, the enhancements would only be a partial solution to improving the existing data retrieval process in the MCA development cycle. The last strategy needs a great effort and suitable for building a new enterprise.

The coupling of a KBES with a DBMS is an attractive alternative for this research, because it does not require modifications to the existing DBMS and still provides the user a tool that contains reasoning capability. Thus, a knowledge-based expert system with domain knowledge in the generation of FOCUS code and further processing on retrieved data can provide a more efficient and effective way of utilizing the CAPCES database.

The tasks of the reasoner include: resolution to the ambiguities left so far; modification on the three query components if necessary; execution of complex reasoning process via procedural processing or rule bases inference; request for generating query language code; and activation of communication module for data retrieval.

To solve ambiguities left, the reasoner may use the following information sources: contexts involved with terms in the query, knowledge about the domain, information about the user.

6.3 Use of Context for Solving Query Ambiguity

The section will talk about how contexts involved with a term are collected, and how they are used to help identify meaning or resolve ambiguities left.

6.3.1 Finding Context

To find contexts involved with a term, the reasoner looks for the contexts that are stored in its Context slot, and associated with the field in its Definition slot, its English name and CAPCES field name, or the term itself. For example, given a definition for the field STATECD which looks like this³⁵:

((STATECD FIELD)

³⁵This is a partial, near representation of the one implemented in Expert-MCA.

((CAPCES-NAME STATECD)
(CAPCES-ALIAS STATE_CD)
(ENGLISH-NAME STATE CODE)
(CONTEXT)))

the reasoner first gets the context in the Context slot (in this case, empty). Then it is added by its English name STATE CODE and CAPCES field name STATECD. Therefore, the contexts collected include CODE, STATE, and STATECD. In general, collected contexts are arranged in the following order: content in its Context slot, the last word of its English name (if the term is a field), rest of its English name, contexts of a field or fields that are referred to in its Definition slot. Such an arrangement for order of contexts for a term can be useful in processing, since a higher priority context indicates a more specific one and may be more often needed.

Given another term CA being defined as a Word/Phrase, as ((CA Word/Phrase) ((DEFINITION STATECD IS 15) (PART-OF-SPEECH NOUN) (CONTEXT LOCATION))), the reasoner will collect the content in its Context slot and the contexts involved with a field in its Definition slot, which is STATECD in this case. Therefore, the contexts found for CA are LOCATION, CODE, STATE, and STATECD. If MAINE is defined as a value for the field STATE, then the contexts involved will be the contexts specified in its Context slot, plus the ones inherited from STATE. In another case, given the term APPROPRIATED being defined as following:

((APPROPRIATED WORD/PHRASE)
(PART-OF-SPEECH VERB) (DEFINITION APPROP_AMT GT 0)
(CONTEXT APPROPRIATE AMOUNT)

the contexts collected are APPROPRIATE, AMOUNT, APPROPRIATED and APPROP_AMT.

Simply by using literal information for each term alone, the language analyzer, as described in the previous chapter, can fail to correctly handle some terms that are associated with obvious key words or contexts around in the same query. For example, it is unable to determine the meaning for MA in the queries:

1. Show the projects in MA with PA over 900;
2. Show the projects in State MA with PA over 900;
3. Show the projects in the MA state with PA over 900;
4. Show the projects in the State of MA with PA over 900.

On another case, it seems very simple to figure out that the requested data item is APPROPRIATED AMOUNT, given the sentence:

5. Show me the amounts that are appropriated for fy 87.

However, the language analyzer cannot derive this meaning either.

To resolve such problems, the reasoner has to use non-literal information, such as a context provided around the unsolved term, use of preposition in context. The following subsections will discuss how the reasoner can resolve these problems by using non-literal information. Problems exemplified in query 1 above will be resolved by using the knowledge that how a preposition can be associated with a context in its use, while those exemplified in the rest queries will be solved by using contextual information from adjacent terms.

6.3.2 Use of Context Driven Reasoning to Resolve Ambiguity

The reasoning mechanisms implicitly implemented in framed-based systems for property inheritance are a special case to context driven reasoning, which is discussed in Section 4.4. Context driven reasoning can be employed to infer meaning association and propagation among objects.

In order to resolve the ambiguity about the term MA in the query "Show me the projects in MA with PA over 900," context driven reasoning can help identify a correct meaning from the two: "STATECD IS 25" and "PRCD IS MA." The process is also shown in Fig. 6-1.

In the domain profile, some propositions about the domain are specified and used to infer new facts in the reasoner. For example, these propositions may be about reasoning knowledge among relational objects. The following two show two propositions stored in Expert-MCA about how reasoning transitivity exists among three relational objects.

Can-be-preceded-by-preposition backward-transit-over is-a-kind-of
Is-a-kind-of backward-transit-over Is-an-instance-of

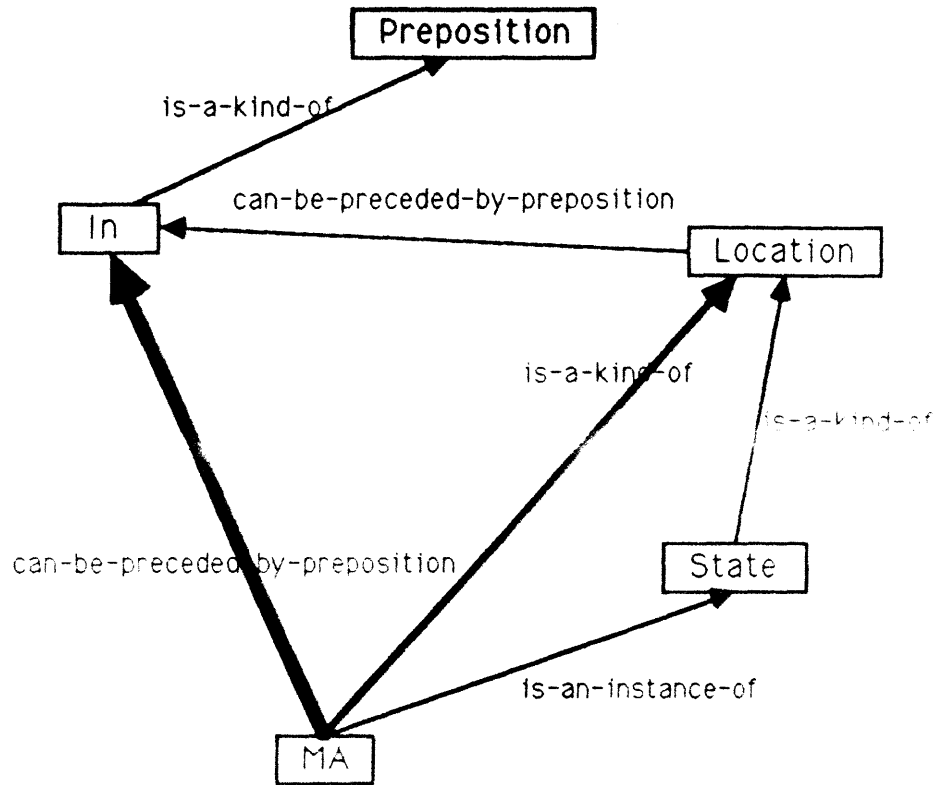
On the other hand, the knowledge about how users use the preposition IN in their queries, and about how one can relate the two objects Location and State can also be specified in the domain profile, as shown as follows:

Location can-be-preceded-by-preposition IN
State is-a-kind-of Location

Now given the facts that the term "MA" follows the term "IN" in the query, one has to check which meaning is more appropriate for "MA." The idea is as follows. If a selected meaning for "MA" is appropriate, then it should be allowed to infer that "MA" can follow "IN." Otherwise one would not use the sequence "IN MA" to indicate the selected meaning.

First, the reasoner has to select one from these two meanings:

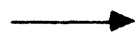



Meaning Association by the context Location and Preposition IN



Given the contextual reasoning knowledge

Can-Be-Preceded-By-Preposition backward-transit-over is-A-kind-Of
is-A-kind-Of backward-transit-over can-be-preceded-by-preposition

Legend

-  Given Relationship
-  Deduced Relationship (first time)
-  Deduced Relationship (second time)
-  Object

is-an-instance-of Name of a Relational Object

Figure 6-1: An Application of Context Driven Reasoning

((MA WORD/PHRASE)
((DEFINITION STATECD IS 25) (PART-OF-SPEECH NOUN) (CONTEXT STATE))
(MA VALUE)
((IN-WHICH-FIELD PRCD) (DEFINITION PRCD IS MA)))

The first meaning selected is "STATECD IS 25" since it is defined as a Word/Phase, which is assigned, in the lexical mapping module, with a higher priority than a Value.

The reasoner will find all possible contexts for this meaning and then makes some instantiation. The contexts collected for this meaning include "state" "code" and "statecd," as described in Section 6.3.1.

The reasoner then asserts propositions that associate the current term with the collected contexts, as shown as follows:

MA is-an-instance-of STATE
MA is-an-instance-of CODE
MA is-an-instance-of STATECD

Next, the reasoner has to collect from the blackboard all related factual data or propositions, including:

- (1) Can-be-preceded-by-preposition backward-transit-over is-a-kind-of
- (2) Is-a-kind-of backward-transit-over Is-an-instance-of
- (3) Location can-be-preceded-by-preposition IN
- (4) State is-a-kind-of Location
- (5) MA is-an-instance-of STATE
- (6) MA is-an-instance-of CODE
- (7) MA is-an-instance-of STATECD

Propositions 1 to 4 above are retrieved from the domain profile, while propositions 5 to 7 are generated based on contexts involved in the selected meaning "STATECD IS 25" for the term "MA." The inference rules used are the three reasoning transitivity rules (Please refer to Section 4.4):

Rule 1.

If
<Object-X> <Relational-Object> <Object-Y>
<Object-Y> <Relational-Object> <Object-Z>
<Relational-Object> is-an-instance-of Transitive-Relation
Then
<Object-X> <Relational-Object> <Object-Z>

Rule 2.

If
<Object-X> <Relational-Object-1> <Object-Y>
<Object-Y> <Relational-Object-2> <Object-Z>
<Relational-Object-1> forward-transit-over <Relational-Object-2>

Then
<Object-X> <Relational-Object-1> <Object-Z>

Rule 3.

If
<Object-X> <Relational-Object-1> <Object-Y>
<Object-Y> <Relational-Object-2> <Object-Z>
<Relational-Object-2> backward-transit-over <Relational-Object-1>
Then
<Object-X> <Relational-Object-2> <Object-Z>

With rule 3 and propositions 2, 4, and 5, the inference engine will infer that

(8) MA is-a-kind-of Location

With rule 3 and propositions 1, 3, and 8, the inference engine again infers that

(9) MA can-be-preceded-by-preposition IN

Therefore the selected meaning is an appropriate one. If the selected meaning cannot be used to infer desired results, another meaning will be similarly processed again.

However, for some cases that two multiple meanings may have the same context, such as amount or code, the idea of using word order in the query is no longer applicable to resolve the ambiguity. In some other cases, there has no enough information that can be applied by using the same kind of inference. For example, the term "MA" in the query "Show me the MA projects for FY 90" will be moved to the screening-condition stack, since it stands for a contextual phrase (although it is unknown which contextual phrase is the correct one at the time it is moved). Nonetheless, its correct meaning cannot be identified either in the semantic analysis module or the reasoner so far³⁶. For this situation, if one of these two meaning is defined with a term type Word/Phrase and another with a term type Value, the reasoner will assume that the one with term type Word/Phrase, which is defined by the user or the designer, will be the default. If the two are all Values, then the user has to specify a default one in user profile.

6.3.3 Using Context From Adjacent Terms

What follows will discuss how the reasoner uses contextual information indicated from adjacent terms to resolve ambiguities. Examples used to illustrate this process include:

1. Show the projects in State MA with PA over 900;

³⁶Because there is no preposition prior to it, one is unable to use the above reasoning process.

2. Show the projects in the MA State with PA over 900;
3. Show the projects in the State of MA with PA over 900.
4. Show me the amounts that are appropriated for fy 87.

As you may have found, the context needed to identify the correct meaning for MA in queries 1 to 3 is from the term STATE. Depending how the user defines STATE in the dictionary of Expert-MCA, it may have many ways to select a meaning for MA from the two "STATECD IS 25" and "PRCD IS MA."

A definition for the term STATE is defined by the system as a Fieldname. The user can also define the term as the following:

```
((STATE WORD/PHRASE)
  ((DEFINITION STATE CONTAINS "?X")
   (PART-OF-SPEECH NOUN)
   (VARIABLE-PART ?X)
   (VARIABLE-LOCATION NEXT)
   (VARIABLE-TYPE STRING)
   (VARIABLE-RANGE N/A))
```

The lexical analysis module will form, as similarly described in Section 5.3. for the term "construction completion percent," a new term "STATE MA" with the meaning "STATE CONTAINS MA." Therefore the language analyzer would not have problem in interpreting the term STATE MA in query 1 above. Similarly, the user can also define STATE differently for coping with such patterns as "?X STATE" and "STATE OF ?X." But this is too tedious, for the user has to define other terms which are similar to STATE in pattern matching as well. This is not the main concern in this subsection, however. Rather, the issue here is how the reasoner can use STATE as a context to help identify a correct meaning for MA.

Now suppose the dictionary contains only one definition for the term STATE, which is a Fieldname. By the time the reasoner starts to work on each of queries 1 to 3, the three modified-screening-condition stack³⁷ for them look like this (neglecting the part for "WITH PA OVER 900"):

For Query 1:
((IN WORD/PHRASE) ((PART-OF-SPEECH PREP)))

³⁷The term THE is a function word, which performs a function role in syntax and semantics. It can be used to help solve ambiguity problems, just as the preposition "IN" does in the previous subsection. However, Expert-MCA does not implement any mechanism to utilize its function role in this respect. Therefore it is seen as a dummy word and removed in the language analyzer. As a result, the term THE in the above stacks for queries 2 and 3 does not appear.

((STATE FIELD)
((CAPCES-NAME STATE) (CAPCES-ALIAS STATE_NAME)
(ENGLISH-NAME STATE NAME) (DEFINITION STATE)))
((MA WORD/PHRASE)
((DEFINITION STATECD IS 25) (PART-OF-SPEECH NOUN)
(CONTEXT STATE))
(MA VALUE)
((IN-WHICH-FIELD PRCD) (DEFINITION PRCD IS MA)))

For Query 2:

((IN WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((MA WORD/PHRASE)
((DEFINITION STATECD IS 25) (PART-OF-SPEECH NOUN)
(CONTEXT STATE))
(MA VALUE)
((IN-WHICH-FIELD PRCD) (DEFINITION PRCD IS MA)))
((STATE FIELD)
((CAPCES-NAME STATE) (CAPCES-ALIAS STATE_NAME)
(ENGLISH-NAME STATE NAME) (DEFINITION STATE)))

For Query 3:

((IN WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((STATE FIELD)
((CAPCES-NAME STATE) (CAPCES-ALIAS STATE_NAME)
(ENGLISH-NAME STATE NAME) (DEFINITION STATE)))
((OF WORD/PHRASE) ((PART-OF-SPEECH PREP)))
((MA WORD/PHRASE)
((DEFINITION STATECD IS 25) (PART-OF-SPEECH NOUN)
(CONTEXT STATE))
(MA VALUE)
((IN-WHICH-FIELD PRCD) (DEFINITION PRCD IS MA)))

The way of using the context STATE to help identify the meaning for MA in the above three queries is described as follows. The reasoner first searches, from left to right, for a term which has multiple meanings. In this case, the term MA will be hit. Next, the reasoner collects the contexts involved with MA for one meaning at a time. If the following steps fail, contexts associated with another meaning will be collected and the following steps will be processed again. At this point, the first set of collected contexts are STATE, CODE, and STATECD. Then the reasoner may check the following condition:

1. Check the case "context-term OF context-instance"³⁸: If the term prior to MA is "OF," then check if the term prior to "OF" is one of the collected

³⁸These three case are corresponding to STATE OF MA, STATE MA, and MA STATE.

contexts. If the check fails then go to the next step. Otherwise it stops and claims the selected meaning for MA is a correct one. At the same time, the two terms prior to MA and the meaning other than the selected one for MA is removed from the stack.

2. Check the case "context-term context-instance": Check if the term prior to MA is one of the collected contexts. If the check fails then go to the next step. Otherwise it stops and claims the selected meaning for MA is a correct one. At the same time, the term prior to MA and the meaning other than the selected one for MA is removed from the stack.
3. Check the case "context-instance context-term": Check if the term next to MA is one of the collected contexts. If the check fails then stops without doing anything to the stack. Otherwise it stops and claims the selected meaning for MA is a correct one. At the same time, the term after to MA and the meaning other than the selected one for MA is removed from the stack.

Having gone through the steps above, the reasoner updates all the three stacks above as follows:

```
((IN WORD/PHRASE) ((PART-OF-SPEECH PREP)))  
((MA WORD/PHRASE)  
((DEFINITION STATECD IS 25) (PART-OF-SPEECH NOUN)  
(CONTEXT STATE))
```

Although the context referred to in the above case is STATE, the algorithm can be used for any context, such as code, cost, etc.

The reasoner also checks to make sure that appropriate items have been collected in requested-item stacks for queries. If a data item in such a stack is not a Field or temporary field³⁹, a neighboring context should be used to refine it. For example, the term AMOUNTS in the query "Show me the amounts that are appropriated for fy 87" is neither a field nor a temporary field.

The following shows how the reasoner finds an appropriate substitute for this term by using contextual information. In this case, the reasoner checks if there is one common context involved with AMOUNTS and the term prior or next to AMOUNTS. If there is one, the reasoner forms a new term and then checks if the new term is a field. A new term is formed by making a combination of two terms, each coming from the two sets of contexts collected for the term and AMOUNTS, respectively. If the new term is a field, then AMOUNTS will be replaced with the new term. Otherwise, the reasoner tries another combination.

³⁹A temporary field should be associated with a Procedure term, or with a Word/phrase that is defined as a statement consisting of two fields connected by an arithmetical operator. Please see Section 4.2.

To be more clear, let us see how it works step by step. By the time Expert-MCA enters the reasoner, the requested-item and screening-condition stacks for this query look like this:

Requested-Item Stack:

```
((AMOUNTS WORD/PHRASE)
 ((DEFINITION AMOUNT) (PART-OF-SPEECH NOUN) (CONTEXT AMOUNT)
 (SYNONYM-OF AMOUNT)))
```

Screening-Condition Stack:

```
((APPROPRIATED WORD/PHRASE)
 ((PART-OF-SPEECH VERB) (DEFINITION APPROP_AMT GT 0)
 (CONTEXT APPROPRIATE AMOUNT)
 (CONTEXTUAL-PHRASE-P TURE)))
(FY 87 WORD/PHRASE)
 ((DEFINITION FY IS 87) (PART-OF-SPEECH NP) (CONTEXT FY)
 (CONTEXTUAL-PHRASE-P TURE)))
```

The term next to AMOUNTS at this point is APPROPRIATED⁴⁰, which is involved with such contexts as APPROPRIATE, AMOUNT, APPROPRIATED and APPROP_AMT. With contexts involved with the term AMOUNTS being AMOUNT and AMOUNTS, the reasoner sequentially checks to see if one of the following terms is a field: APPROPRIATE AMOUNT, AMOUNT AMOUNT, APPROPRIATED AMOUNT, APPROP_AMT AMOUNT, APPROPRIATE AMOUNTS, AMOUNT AMOUNTS, APPROPRIATED AMOUNTS, and APPROP_AMT AMOUNTS. As a result, the term APPROPRIATED AMOUNT is found as Fieldname. The reasoner then replace AMOUNTS in the requested stack with the new term APPROPRIATED AMOUNT.

6.4 Use of User and Domain Profile Information

The reasoner also refines contents of the three query components by using information about the user and the domain and by activating the Profile Knowledge Source, a rule base about how information or data items should be used to add (or delete) to query components by considering, for example, user job characteristics or responsibilities, or regulations about the application domain. Specifically, the rule base contains knowledge on how to specify default printing data items, default sorting data items, and default screening conditions given

⁴⁰At this point, the terms THAT and ARE are removed in the semantic analysis module, since they are treated as dummy words. Please see Section 5.5.

the role and responsibilities of the user and the context of the sentence. Currently, Expert-MCA does not implement too many of such rules. The rule base can be expanded or modified (in Profile Session) as the user sees a need.

Some of rules are explained as follows. Set-Rpt-Rule is used to set up default printing items according to the agency of the user. Set-Printer-Rule is used to set up data items for requested and sorting information, according to user agency and contexts of the query. Set-Download-Select-Rule is used to specify screening conditions in downloading a subset of topical database from the mainframe. A similar concept can also be used for the screening conditions by considering the user's information and domain's practices.

For example, Set-Printer-Rule looks like this:

Rule name: MACOM-1

```
IF
  USER IS-A MACOM
  CONTEXT IS-A AMOUNT
THEN
  PRINTER CONTAINS PROJECT DESCRIPTION AND PROGRAM AMOUNT
  SORTER CONTAINS MAJOR COMMAND AND STATE CODE
```

Rule name: DAEN-PRINTER

```
IF
  USER IS-A DAEN
  CONTEXT IS-A AMOUNT
THEN
  PRINTER CONTAINS PROGRAM AMOUNT AND APPROPRIATED AMOUNT
  SORTER CONTAINS FISCAL YEAR AND DISTRICT NAME
```

Rule name: MACOM-2

```
IF
  USER IS-A MACOM
  CONTEXT IS-A DATE
  CONTEXT IS-A DESIGN
THEN
  PRINTER CONTAINS DESIGN START DATE AND DESIGN PERCENT
  AND DESIGN COMPLETION DATE
  SORTER CONTAINS FISCAL YEAR AND STATE
```

The rule MACOM-1 says that if the user's agency is a major command and the query is involved with the context amount, then the requested data items should contain such fields as PROJECT DESCRIPTION and PROGRAM AMOUNT; the sorting data items should contain such fields as MAJOR COMMAND and STATE CODE. The rule DAEN-PRINTER says that if the user's agency is the Department of the Army, Engineering Office (including

the Office of Chief Engineers and Assistant to Chief Engineers Office) and the query is involved with the context amount, then the requested data items should contain such fields as PROGRAM AMOUNT and APPROPRIATED AMOUNT; the sorting data items should contain such fields as FISCAL YEAR and DISTRICT NAME. The rule MACOM-2 says that if the user's agency is a Major Command and the query is involved with the context date and design, then the requested data items should contain such fields as DESIGN START DATE and DESIGN PERCENT and DESIGN COMPLETION DATE; the sorting data items should contain such fields as FISCAL YEAR and STATE.

In Expert-MCA, each of such rule set contains several production rules and is associated with a term with a term type Rule, such as Set-Printer-Rule, or Set-Download-Select-Rule. These rule sets (or Rule terms) are defined and customized by the user. To tell the reasoner which rule sets are needed, the user has to fill in their names in the Domain-Rule-Sets slot at running a Domain Profile Session⁴¹.

On the other hand, adding data items to the query components can be personalized. The user may have a specific interest in some data items for a short period or for a non-routine work. This requires that the system be flexible for him or her to specify more dominating information about requested data items, sorting data items, or screening conditions. In Expert-MCA, the user can specify such information in the Default-Printed-Items, Default-Sorting-Items, and Default-Screening-Conditions slots, respectively, at running a User Profile Session.

This default information can be very useful in refining the query components. For example, a user, who is working on a specific group of projects in fiscal year 1989, forgot to specify the fiscal year in his or her query. By having the default range of time period for the projects from the user's profile, the reasoner can add additional screening conditions for data retrieval. This might save a lot of time and efforts without retrieving information that is not needed.

On another case, the user often asks "What are the projects" with some following conditions in a query. However, in the CAPCES database, there is no such a field called "project" or "projects." Rather it means a collection of some fields, such as PROJECT DESCRIPTION, PROJECT KEY NUMBER, and PROGRAM AMOUNT. Although the

⁴¹The Profile Session contains two branching sessions: Domain Profile Session and User Profile Session. Please refer to Chapter 8.

user can define the term as some specific fields, such a definition for the term may add complication to analysis for queries. This is because the term **PROJECTS** (or **project**) can be a dummy word in many locations in a query. A better approach, which is implemented in Expert-MCA, is to allow the user to specify those fields in user profile as default printing items. The reasoner then just removes the term **PROJECTS** (or **project**) from the query components and add the default printing items to the query components.

The user profile can also store a default meaning for a term that is unable, or not easy, to determine a correct one among its several possible meanings. For example, if a term which is a value for multiple fields appears in a query and there is no contextual term around in the query, Expert-MCA must check with the user profile for a default meaning, or ask the user to select one in the middle of a Query Session.

6.5 Execution of Tasks Triggered by Terms

The reasoner next checks the terms to see if they invoke any procedure or rule. At the same time it also removes from the three query stacks any term which is a function word, such as prepositions and conjunctive (**AND**, **OR**), since the reasoner no longer uses function words for helping identify the meaning of the query. If any of the terms in the query refers to a procedure or a rule, the reasoner activates an appropriate inference engine to reason new facts, based on the statements in Procedures or Rules. Such facts then are posted onto the blackboard. A Procedure often needs other term types such as Fieldnames or Fieldvalues to do arithmetic calculations or logical analyses. The reasoner searches for this information from the blackboard. If such information does not exist on the blackboard, the reasoner will request it from the user.

In a rule base environment, an inference engine is a computer program that can infer information symbolically based on a set of rules and an initial set of working facts. A working memory is used to store the facts that include the input working facts initially and will be updated after rules are applied. An inference engine uses some control mechanisms to guide how repetitively matches the data in the working memory with the rules provided and to update the working memory. Although the two algorithms forward-chaining and backward-chaining are usually used in an inference engine, Expert-MCA primarily applies the forward-chaining algorithm to infer new facts because the number of given facts that are conveyed in the input query is quite small.

In addition to an inference engine for inferencing production rules, Expert-MCA also implements a procedure processor for processing procedures that are triggered by some terms in the query. When procedures are detected by the reasoner, they will be sent to the procedure processor. Basically, the procedure processor performs as a miniature of compiler and loader. It first compiles the definition statements for the procedure and converts these statements into equivalent expressions in Lisp functions. The procedure processor then evaluates the Lisp expressions to actually execute tasks as specified in the procedure.

6.5.1 Rule Base Inference Used for Answering a Query

The system can also be used to store general knowledge, in the form of rules and procedures, about the tasks in the MCA development cycle. Such knowledge can be encoded from the regulations and practices of the organization. This subsection discusses how the reasoner processes a procedure which in turn calls a rule base, while the next subsection discusses how it processes a procedure which calls other two procedures.

A user may directly query such knowledge. For example, a user might want to know the latest date that he may submit his or her projects to OCE (the Office of Chief Engineers) and still have his project included in a program year. For such a query, the system functions more like a simple expert system which uses domain knowledge called up with a natural language interface.

The user may give a query such as:

WHEN IS THE LATEST DATE TO SUBMIT PROJECTS TO OCE FOR FY 91?

By the time the reasoner starts working, the stacks created by the language analyzer look like this:

Requested-Item Stack:

```
((LATEST DATE WORD/PHRASE)
 ((DEFINITION LATEST DATE) (PART-OF-SPEECH NP)
 (CONTEXT DATE-TYPE TIME)))
```

Screening-Condition Stack:

```
((SUBMIT PROJECTS TO OCE WORD/PHRASE)42
 ((DEFINITION SUBMIT PROJECTS TO OCE) (PART-OF-SPEECH VP)))
 ((FY 91 WORD/PHRASE)
```

⁴²The term SUBMIT is defined as a Word/phrase which expects to bind other words. What we see here is a result after the lexical analysis. However, it is also defined as a Procedure with a definition shown below.

((DEFINITION FY IS 91) (PART-OF-SPEECH NP) (CONTEXT FY)))

At this point, the reasoner checks if any term in the Definition slots for the query stacks left refers to a Procedure or Rule. Because a procedure SUBMIT is found in the Definition slot, the reasoner will ask the procedure processor to process it.

In this example, the term SUBMIT is also defined as a procedure which develops a process for determining a date for a type of submittal. Its definition is as follows:

```
SUBMIT PROC      ;declare a knowledge procedure SUBMIT
PART-OF-SPEECH VERB
DECL FACT X Y Z U ;declare X, Y, Z, U as facts
GET X OBJECT     ;get a fact bound to a context OBJECT
GET Y AGENCY     ;get a fact bound to a context AGENCY
GET Z DATE-TYPE ;get a fact bound to a context DATE-TYPE
U = DETERMINE DATE-TYPE USING SUBMIT-RULES
                ; call the rule SUBMIT-RULES with an input
OUTPUT U
```

The SUBMIT procedure specifies the need to find facts associated with an object, an agency, and a date-type before activating a rule base "SUBMIT-RULES". The reasoner searches the user query from the blackboard and finds out the following facts: the object for the submittal is "projects"; the receiving agency for the submittal is "OCE"; the date-type for the submittal is "latest"; and the submitting agency by default is the user's agency, which is found from the user profile, in this case MACOM (for Major Army Command). Given these facts in conjunction with FISCAL YEAR IS 91, the knowledge rule "SUBMIT-RULES" is fired.

The SUBMIT-RULES rule base, partially shown below, contains knowledge of the paperwork flow events in the construction program cycle, including submission dates and constraints.

```
((SUBMIT-RULES RULE)
....
....
(RULE EVENT-16-1
 (IF (AGENCY IS-A OCE)
      (USER IS-A MACOM)
      (OBJECT IS-A PROJECTS))
 (THEN (EVENT IS-A EVENT-16)))
....
....
(RULE TIME-EVENT-16
 (IF (EVENT IS-A EVENT-16)
      (DATE-TYPE IS-A LATEST)
      (FY IS ?X))
 (THEN (THE LATEST DATE IS JUNE 30 (1900 + (?X - 2)))))
....
....
)
```

The inference engine fires rules which relate the facts to dates. In this example, an event, Event-16, is inferred as associated with a submission of projects from a MACOM to OCE. This rule calculated the late date for Event-16. The variable ?X has been bound to FY for programmed year and has a value of 91 in this example. Then the latest dates for the submission is June 30 1989. Therefore the fact "THE LATEST DATE TO SUBMIT IS JUNE 30 1989" is returned from the rule base SUBMIT-RULES to the calling knowledge procedure SUBMIT, which in turn outputs the fact.

6.5.2 Procedural Processing Used for Answering a Query

Another user might be interested in determining the amount of funds to be allocated for the design of the projects in Massachusetts within his district office in fiscal year 1988. CAPCES does not store such data. This user might ask an expert how design budgets are determined. The expert might tell him that the amount of the design funds in one particular year can be approximated as 4% of the program amount of the projects for the following year which have less than 25% of their design completed, plus 5% of the program amount of the projects programmed for two years in the future for which the design has yet to start.

The user would like to query the system by saying:

WHAT ARE THE DESIGN FUNDS FOR MA PROJECTS IN FY 88?

In Expert-MCA, DESIGN FUNDS must be defined, either by the user if he/she has some programming skills, or by a programmer using the above heuristics. The user can

define the three terms; design funds, design-fund1, and design-fund2, each as a procedure. Once they are stored in the system, they become part of the system's knowledge and can be used again and by others. The definition of these three terms might look like this:

```
;;This column is for comments.
DESIGN-FUNDS PROC      ;declare a knowledge procedure DESIGN-FUNDS
PART-OF-SPEECH NOUN   ;
DECL FACT X            ;declare X as a fact
DECL VAR X1 X2        ;declare X1 X2 as numerical variables
DECL FIELD F1 F2 F3   ;declare F1 F2 F3 as temporary fields
IF SCREENFIELDS NOT CONTAIN FY THEN ASK FY
    ;if screening-condition stack does not
    ;contain the field FY, ask the user to input
GET X FY               ;get a fact bound to a field FISCAL YEAR
X1 = X + 1
X2 = X + 2
F1 = CALL DESIGN-FUND1 (X1) ;call procedure DESIGN-FUND1
F2 = CALL DESIGN-FUND2 (X2) ;call procedure DESIGN-FUND2
F3 = F1 + F2
REMOVE FY FROM SCREENFIELDS ;remove FY from screening fields
APPEND DISTRICT TO SORTFIELDS ;append DISTRICT to sorting fields
OUTPUT SUM F3 BY SORTFIELDS FOR SCREENFIELDS ;output a sum
```

The procedure DESIGN-FUNDS starts by declaring types of information needed, including facts, variables, temporary field names, and tables. Then, it checks whether a necessary piece of information was supplied in the query. If not, it will ask for the information from the user. Arithmetical computations and other processing such as evaluating other procedures will be executed sequentially. Database query conditions are modified by removing the query selection condition for fiscal year and adding a sorting condition. Lastly, the report's contents is specified by requesting a summation of a calculated field with sorting and selection criteria.

```
DESIGN-FUND1 PROC      ;declare a knowledge procedure DESIGN-FUNDS
PART-OF-SPEECH NOUN
DECL FIELD Y
DECL VAR X1
GET X1 INPUT          ;get the variable X1 from the calling procedure
Y = IF FY IS X1 AND DESIGN PERCENT GT D00 AND DESIGN PERCENT
    LT D25 THEN Y = 0.04 * PROGRAM AMOUNT
WHEN Y GT 0
OUTPUT Y
```

```
DESIGN-FUND2 PROC      ;declare a knowledge procedure DESIGN-FUND1
PART-OF-SPEECH NOUN
DECL FIELD Y
DECL VAR X2
GET X2 INPUT          ;get the variable X1 from the calling procedure
Y = IF FY IS X2 AND DESIGN PERCENT EQ D00
  THEN Y = 0.05 * PROGRAM AMOUNT
WHEN Y GT 0
OUTPUT Y
```

The procedure DESIGN-FUND1 contains conditions for selecting the projects with design completion less than 25% for a specific fiscal year, and calculating 4% of the program amounts of these projects as the procedure's output. Similarly, the procedure DESIGN-FUND2 contains instructions for selecting the projects with design yet to start in a specific fiscal year, which when called is the following, and calculating that 5% of the program amounts of these projects be taken as the procedure's output.

When the reasoner starts to screen the blackboard for a procedure or rule in the query "What are the design funds for MA projects in FY 88." the blackboard has the stacks which look this:

Requested-Item Stack:

```
((DESIGN FUNDS PROC)
((PART-OF-SPEECH NOUN) (DECL FACT X) (DECL VAR X1 X2)
... ..
(DEFINITION DESIGN-FUNDS) (SYNONYM-OF DESIGN-FUNDS)))
```

Screening-Condition Stack:

```
((MA WORD/PHRASE)
((DEFINITION STATECD IS 25)
(PART-OF-SPEECH NOUN)
(CONTEXTUAL-PHRASE-P TRUE)))
((FY 89 WORD/PHRASE)
((DEFINITION FY IS 89)
(PART-OF-SPEECH NP)
(CONTEXTUAL-PHRASE-P TRUE)))
```

The reasoner now detects that a procedure DESIGN-FUNDS is referred to in the query. It then asks the procedure processor to execute the procedure DESIGN-FUNDS. During its processing, it discovers that it needs two other procedures, DESIGN-FUND1 and DESIGN-FUND2. It creates two other environments for executing the two procedures DESIGN-FUND1 and DESIGN-FUND2. After finishing each of the two procedures, processing continues in DESIGN-FUNDS.

By integrating the results of these three procedures, the query generator is able to define the needed temporary fields which are incorporated in the FOCUS code. They are used to the construct the following calculation section for FOCUS:

```
DEFINE FILE PMMFILE
DESIGN-FUND1 = IF (FY IS '89' AND DES_PERCENT GT 'D00'
AND DES_PERCENT LT 'D25') THEN 0.04 * PROG_AMT ;
DESIGN-FUND2 = IF (FY IS '90' AND DES_PERCENT EQ 'D00')
THEN 0.05 * PROG_AMT ;
DESIGN-FUNDS = DESIGN-FUND1 + DESIGN-FUND2 ;
TEST-D-F = IF DESIGN-FUND1 GT 0 THEN 'Y'
ELSE IF DESIGN-FUND2 GT 0 THEN 'Y' ELSE 'N' ;
END
```

The query generator then will generate the main part of FOCUS code as:

```
TABLE FILE PMMFILE
HEADING CENTER
" REPORT ON DESIGN FUNDS"
" </2 "
" <20 MADE BY SYSTEM <60 ON 10-30-88 "
SUM
DESIGN-FUNDS
BY DIST_NAME
IF STATECD IS '25'
IF TEST-D-F IS 'Y'
END
```

6.6 Pre-Access Processing and Post-Access Processing

To answer a query, one may need a complex process, via specification in Procedures or Rules, that operates on information both prior to and after data retrieval from the database. Further processing, including post-access (or post-retrieval) processing, is only carried out through the remainder of active Procedures. The primary objective of pre-access processing is to translate information about the query to its corresponding FOCUS code. The next chapter will discuss how this is accomplished.

Most procedures are used to create temporary fields, such as DESIGN-FUNDS, DESIGN-FUND1, and DESIGN-FUND2 above. The reasoner can perform the calculations for the temporary fields within FOCUS by sending FOCUS code as a prefix to a report request. It may also import the raw data and perform the calculations in post-access processing. Because of the speed of the mainframe and convenience, if performing calculations and setting conditions can be processed in the database system and post-access

processing is not required, the reasoner prepares and sends pre-access processing instructions to the query generator.

However, there are queries which can only be solved by using post-access processing. The reasoner must scan procedures and determine if the process described in the procedures can be processed completely within a pre-access processing, and thus just reports the received information to the user. As illustrated below, a need for a post-access processing can be detected by looking at the sequences of use of needed fields or factual data in procedures or rules used. If any operation specified in a procedure cannot continue without providing data retrieved from the database, then the reasoner has to proceed in a post-access processing⁴³

Now let us consider the following query: "Show me the design performance for FY 87 projects by designers." Assume the inquirer is working as a manager in a design agency and wants to know how each of his or her design employees performs in design. Also the following assumptions are made in order to process an evaluation about design performance: each project has a project ID and jointly designed by many designers; a bad designer tends to have bad design performance with the projects he or she is involved; one may know who is responsible for project's delay by looking at the patterns of design performance with all the projects associated with designers.

The reasoner detects that data item DESIGN PERFORMANCE collected in the requested-item stack and then will inform the procedure processor to execute the procedure. Thus the definition for the procedure will be retrieved, being shown as below:

```
DESIGN-PERFORMANCE PROC ;Declare a procedure
DECL VAR ID X ;Declare two variables ID and X
GET ID DESIGNER-ID ;Find an ID value by looking for
; the context DESIGNER-ID
FOR-EACH ID DO ;For each ID, do the following:
X = AVERAGE (DESIGN-PERFORMANCE-INDEX)
;Assign X an average over design-performance-index involved
;with the current ID. Design-performance-index is another procedure.
OUTPUT X ID AS "DESIGN PERFORMANCE" "DESIGNER ID"
;Output the values for the two columns X and ID
END-DO ;End of FOR-EACH loop, the output is a table.
```

The procedure processor then compiles each of the above statements into equivalent Lisp

⁴³Since post-access processing is much more complicated than pre-access processing, it has yet been fully implemented in the prototype system Expert-MCA. However, a conceptual idea about how to process a post-access processing is illustrated in the DESIGN PERFORMANCE case.

expression. In the middle of the compiling process, it recognizes that DESIGN-PERFORMANCE-INDEX is another procedure. Therefore, the procedure processor continues to compile the procedure DESIGN-PERFORMANCE-INDEX. As such, the procedure processor records all procedures and rules, together with their variables and detected fields.

The definition for DESIGN-PERFORMANCE-INDEX is show as below:

```
DESIGN-PERFORMANCE-INDEX PROC ;Declare a procedure
DECL VAR X Y Z INDEX ;Declare variables X, Y, Z, and INDEX
GET X PROJECT_ID ;Assign X a value associated with Project_ID
FOR-EACH X DO ;For each X do the following:
  Y = DETERMINE EXPECTED-DESIGN-DURATION
    USING EXPECTED-DESIGN-DURATION-RULES
    ;Assign Y a value that is determined for expected design duration
    ; by using a rule base EXPECTED-DESIGN-DURATION-RULES.
  Z = Retrieve Actual-Duration
    ;Retrieve a value for term associated with the current X
    ;(or the project ID in question).
  INDEX = Z / Y
  OUTPUT = INDEX Z Y X AS "DESIGN PERFORMANCE INDEX" "..." "..." "..."
END-DO ;Finish the For-Each loop, and output a table with above four columns.
```

A rule base EXPECTED-DESIGN-DURATION-RULES and a term Actual-Duration are needed in the above procedure. Definitions for them are shown as below:

```
EXPECTED-DESIGN-DURATION-RULES RULE ;Declare a rule
RULE-1
```

```
RULE-8
```

```
IF CATCD3 FROM 100 TO 199 ;If project's category code is from 100 to 199
  PROG_AMT FORM 200 TO 300 ;and program amount is from 200 to 300 thousands
THEN EXPECTED-DESIGN-DURATION IS 60
  ;then expected design duration is 60 days.
```

```
RULE-9
```

```
...
...
```

```
ACTUAL-DESIGN-DURATION WORD/PHRASE ;This is a Word/phrase.
DEFINITION DESIGN COMPLETION DATE - DESIGN START DATE
  ;Difference of two fields
PART-OF-SPEECH NOUN
```

```
....
```

The procedure processor finally reports to the reasoner that a post-access processing is needed and the following data items are required for its further processing: DESIGNER_ID, PROJECT_ID, CATCD3 (for project category code with 3 digits), PROG_AMT (for program amount), ACTUAL-DESIGN-DURATION (a temporary field associated with two fields DESIGN COMPLETION DATE AND DESIGN START DATE).

Upon receiving the report from the procedure processor, including cases for both pre-access processing and post-access processing, the reasoner must request the query language generator to compose the database query, send it to the DBMS, and download the results. If the case needs a post-access processing, the returning results will send to the procedure processor for continuing its process.

As illustrated in the above example, statements specified within a Procedure is often used to infer additional information from the retrieved data in order to obtain a proper solution. The post-access processing can range from a simple algorithmical operation to a complicated pattern search, or from a diagnosis of problems to a forecast of trends.

Once the query generator produces a set of FOCUS statements, the reasoner has to exploit either the local databases stored in the personal computer or the mainframe database in Dallas, Texas. If data items included in the FOCUS code exist in a local database, the data retrieval will be done locally by sending the FOCUS code to PC/FOCUS, a PC version of FOCUS database management system. Otherwise the reasoner will send the FOCUS code to the mainframe, together with necessary communication parametric information. Such information, for example, includes a username for entering the mainframe, a telephone number connected to the mainframe, and values for a set of communication parameters, such as baud and port. The reasoner then is able to trigger a telecommunication process to the mainframe.

Chapter 7

Query Language Generator and Knowledge Acquisition

7.1 Query Language Generator

The query language generator is a module that translates the internal representation for the input sentence into the target query language FOCUS. Before entering this module, the internal representation has been refined through previous processing modules. The query language generator simply extracts the meaning expression that is stored in the Definition slots of the terms in the three refined groups: requested items, screening conditions, and sorting items. It then follows the syntax for FOCUS to construct FOCUS query expression.

7.1.1 Features of the Query Language FOCUS

FOCUS is a product of Information Builders, Inc [79]. It is an information control system, with facilities for describing files, for entering, changing, and deleting records in the files, and for preparing reports from the information in the files. This study is concerned with its report request language only. FOCUS files are arranged in a structure in which different segments of data are related to each other in a hierarchical parent-to-child relationship. Although FOCUS query language is used to interact with hierarchical data models, it has been carefully designed to generate reports efficiently and easily. It is claimed that little or no knowledge of file structures is needed to use the report request language.

To prepare reports, users have to program their requests in terms of FOCUS request language, or so-called query language. Such request programs specify the directions for which records are to be retrieved, what calculations are to be performed, how the lines of the resulting reports are to be sorted, and how the report is to be formatted on display. A report program may consist of report request statements, report formatting statements, report pre-processor statements.

A tabular report is requested by a TABLE command statement, which consists of several phrases: such as TABLE-phrase, verb-phrase, sorting-phrase, screening-phrase, and End-phrase. A detailed description about the syntax of the TABLE command statement is given in Appendix D.1. The TABLE command statement looks like this:

**TABLE FILE <filename>
<verb-phrase>
[<sorting-phrase>]
[<screening-phrase>]
<end-phrase>**

where [<phrase>] means <phrase> is optional.

<Filename> is the file from which data is to be retrieved. <End-phrase> can be a single command, such as END, RUN, or QUIT. A verb in FOCUS is a word of action that is performed on the fields which are its objects in a verb-phrase. A verb-phrase looks like this: Verb fieldname and fieldname and fieldname For example, PRINT PROG_AMT (for program amount) AND APPROP_AMT (for appropriated amount) is a verb-phrase. The verbs used are LIST, PRINT, COUNT, SUM, ADD, and WRITE. LIST is to list the records one on a line, and to number them. PRINT is to list the records one on a line, but not to number them. COUNT is to count the number of occurrences. SUM is to write a record in a line which may contain other records, adding computational fields together, overwriting alphanumeric fields. The use of ADD or WRITE is the same as SUM. The conjunction AND between two fieldnames are optional.

The lines of a report page can be sorted by a field, including temporarily defined fields (see below), or a list of fields. The report can also be sorted across the columns of a page by a field or a list of fields (but not more than 5 fields). A matrix type of reports can be produced by sorting fields in rows, using the command BY, and columns, using the command ACROSS. The default sorting sequence is low to high, A to Z, or 0 to 9. The format of a sorting statement looks like this: BY fieldname or ACROSS fieldname. To produce a tabular report with a hierarchy in sorted fields, we can write a sorting-phrase consisting of several sorting fields sequentially. For example, if we have the sorting statement:

**BY STATE
BY FISCAL_YEAR**

then the data in the target tabular report will be categorized (sorted) by the sequence of STATE, with a subcategory in terms of FISCAL YEAR within each of the occurrences of STATE.

Screening conditions, as specified in screen-phrase, are used to limit the acceptance of individual records to the ones which past the tests. These are the records which are then sorted, accumulated, and processed into the output format of the finished report. If a TABLE

command statement does not contain screening conditions, every record of requested fields in the data file will be considered acceptable and hence retrieved. A screening condition looks like this: IF fieldname <relation> <value>, where <relation> stands for a logical relation such as IS, EQ (for equality), NE (for inequality), GT, GE, LT, LE, TO, FROM (for a range); <value> stands for a data instance or a number. A screening-phrase may consists of multiple screening conditions. For example, the following is a valid screening-phrase:

```
IF FY GE 90
IF STATE IS MASSACHUSETTS
```

where

FY stands for fiscal year, and GE stands for greater than.

In FOCUS, temporary data fields can be defined as arithmetical or logical combinations of real fields or other temporary data fields. To define temporary data fields needs to issue DEFINE command statements, each having the format as follows:

```
DEFINE FILE filename
temporary-field-1/format-1 = expression-1 ;
temporary-field-2/format-2 = expression-2 ;
...
...
END
```

A more detailed description of the syntax of the DEFINE command statements is given in Appendix D.2. Once a temporary data field is defined, it can be used later in report requests in exactly the same way as a real field is used, such as in verb-phrase, sorting-phrase, or screening-phrase. In other words, the computed or derived data fields by using the DEFINE command statements appear to users (or internally to report generator of FOCUS) as if they were actual fields.

The use of such DEFINE command statements is the major part of what we called report pre-processor statements previously. The formatting statements are used to echo text, such as heading, and footing, for supplemental purposes. They are composed of statements which are described within the symbols " ", or are led by some special commands such as HEADING, ON, FOOTING.

7.1.2 Composition of FOCUS Code in Expert-MCA

Since Expert-MCA has grouped, in the language analyzer and the reasoner, user's input queries into the three components: requested data items (or printing items), screening conditions, and sorting data items, it is fairly straightforward to compose a TABLE command statement. In theory the internal representation for such three components is nothing to do with any database query language, because any database query must contain the information about the three components⁴⁴, and hence the query can be transformed into a single set of information in terms of an internal representation, as long as a set of syntax and semantic for such a transformation is followed.

When the FOCUS query generator⁴⁵ is requested to compose FOCUS code, it retrieves the information about the three components from the blackboard and then applies a set of rules to transform the information into target FOCUS code⁴⁶. Such rules can be described in concept as follows:

Rule 1.

If Action-In-Query⁴⁷ is <verb>
Then Report-Verb has-meaning-or-value <verb>

Rule 2.

If <term> is-a-member-of Requested-Data-Item
Then <field> is-derived-by-finding-FOCUS-field-from <term>
FOCUS-Requested-Field is-pushed-with-attribute-value
Has-Meaning-Or-Value <field>

Rule 3.

If FOCUS-Requested-Field has-meaning-or-value <fields>
Report-Verb has-meaning-or-value <verb>
Then <phrase-value> is-concatenated-by <verb> <fields>
Verb-Phrase is-pushed-with-attribute-value
Has-Meaning-Or-Value <phrase-value>

⁴⁴If screening conditions and sorting data items are not provided in the query, it may imply some default information about them. In Expert-MCA, they can be specified in user profile.

⁴⁵This is the generator designed as a module of Expert-MCA to generate FOCUS code. Please do not get confused with the report generator of FOCUS.

⁴⁶The transformation is implemented in Lisp procedures. The following rules presented here are only for explanation purpose.

⁴⁷Please see Section 5.5.

Rule 4.

**If <term> is-a-member-of Sorting-Data-Item
Then <field> is-derived-by-finding-FOCUS-field-from <term>
<field> is-a-value-of FOCUS-Sorting-Field**

Rule 5.

**If <field> is-a-value-of FOCUS-Sorting-Field
Then <phrase-value> is-concatenated-by BY <field>
Sorting-Phrase is-pushed-with-attribute-value
Has-Meaning-Or-Value <phrase-value>**

Rule 6.

**If <contextual-phrase> is-a-member-of Screening-Conditions
Then <cond> is-derived-by-finding-FOCUS-screen-from <contextual-phrase>
<cond> is-a-value-of FOCUS-Screening-Conditions**

Rule 7.

**If <cond> is-a-value-of FOCUS-Screening-Conditions
Then <phrase-value> is-concatenated-by IF <cond>
Screening-Phrase is-pushed-with-attribute-value
Has-Meaning-Or-Value <phrase-value>**

Rule 8.

**IF Verb-Phrase has-meaning-or-value <verb-phrase-value>
Sorting-Phrase has-meaning-or-value <sorting-phrase-value>
Screening-Phrase has-meaning-or-value <screening-phrase-value>
Then <phrase-value> is-concatenated-by
TABLE FILE PMMFILE
<verb-phrase-value>
<sorting-phrase-value>
<screening-phrase-value>
END
TABLE-Command-Statement is-pushed-with-attribute-value
Has-Meaning-Or-Value <phrase-value>**

Note:

1. <content> stands for a variable object (named content), which is to be replaced by facts when such rules are used.
2. PMMFILE is the file from which Expert-MCA retrieves data.

The relational object **is-derived-by-finding-FOCUS-field-from** in rules 2 and 4, or **is-derived-by-finding-FOCUS-screen-from** serves as a derivational object, which is used to derive its preceding objects by taking its following object (or objects) as its input. Similarly, the object **is-concatenated-by** is also a derivational object. As mentioned in Section 4.3, the use of an object which can be a relational object and a derivational object is two-fold. It can be used in context driven reasoning to relate objects. It can also be used to derive, by activating an

associated Lisp function, a value for the object preceding it. That how such objects are used is solely depending on the environment in which the objects are activated.

To illustrate, let us consider the query: Show the current working estimate for fy 90 projects by size. Prior to entering the FOCUS query generator, the blackboard contains such information as:

1. Action in query: PRINT⁴⁸
2. Requested data items: ((CURRENT WORKING ESTIMATE) (meaning: CWE) ...)
3. Sorting data items: ((SIZE) (meaning: PROG_AMT) ...)
4. Screening conditions: ((FY 90) (meaning: (is FY 90) ...)

As shown above, the internal representation is expressed as a combination of axioms in formal logic form. Being defined as a synonym of the field CWE, the term Current Working Estimate is transformed into a predicate expression: (meaning: CWE). Because the database with which Expert-MCA considers to interact is solely the CAPCES database, Expert-MCA directly transforms definition of user's terms into fieldnames of the CAPCES database.

Using the information above, the FOCUS query generator has to transform it into the statements:

5. Action-In-Query is PRINT
6. CWE is-a-member-of Requested-Data-Item
7. Prog_Amt is-a-member-of Sorting-Data-Item
8. (FY is 90) is-a-member-of Screening-Conditions

Given statements 5. and 6. and rules 1, 2, and 3, we can infer that

9. Report-Verb has-meaning-or-value "PRINT"
10. Verb-Phrase has-meaning-or-value "PRINT CWE"

Note that the value "CWE," being instantiated by the derivational object is-derived-by-finding-FOCUS-field-from in Rule 2, is pushed into the attribute Has-Meaning-Or-Value of the object FOCUS-Requested-Field, while the value "PRINT CWE," being instantiated by the derivational object is-concatenated-by in rule 3, is pushed into the attribute Has-Meaning-Or-Value of the object Verb-Phrase. Given statements 7. and 8, and rules 4, 5, 6, and 7, we can similarly infer

11. Sorting-Phrase has-meaning-or-value "BY PROG_AMT"
12. Screening-Phrase has-meaning-or-value "IF FY IS 90"

Finally, given statements 10, 11 and 12, and rule 8, we have

TABLE-Command-Phrase has-meaning-or-value

⁴⁸This is a default set by the semantic analysis module. Please see Section 5.5.

```
" TABLE FILE PMMFILE
  PRINT CWE
  BY PROG_AMT
  IF FY IS 90
  END "
```

The symbols used internally in the meaning: slot, as shown in statements 1, 2, 3, or 4, are not necessarily transformed from user's definition into field names of a specific database. While multiple target databases are used, such transformation to target field names should be deferred until it is necessary in a query language generator. It can be done by two steps. For each term in user's queries, we first have to generate a symbol standing for its meaning internally, by applying a simple algorithm, such as to form it by replacing the blank spaces " " between words in a term with the underscore mark "_" . Expert-MCA employes such symbols internally, mostly in a formal logic format. Next, the query generator has to consult a table which specifies how fieldnames of different databases are associated with the symbols being defined in the first step.

The user often needs to request information that is not directly stored in the CAPCES database. In other words, Expert-MCA should allow users to express derived-items, or temporary data fields as opposed to real data fields in the database. For such temporary data fields, Expert-MCA first generates their corresponding DEFINE command statements and then puts the statements prior to a TABLE command statement to form a complete set of FOCUS query code. Once a temporary data field is defined, it is used exactly the same as a real field in a following TABLE command statement.

Again, let us consider an example. "Show the projects in fy 90 by overrun cost" is a query to print a report with data sorted by the data item overrun cost, in a low-to-high order. However, the term Overrun Cost is not a field name of the CAPCES database. Users can define it, for example, as: Current Working Estimate - Program Amount if Current Working Estimate is greater than Program Amount. Such a meaning expression is transformed, in the lexical mapping module, into an internal expression led by the symbol meaning:, as shown below:

```
((Overrun Cost word/phrase)
 (part-of-speech: noun)
 (definition: Current Working Estimate - Program Amount
  if Current Working Estimate is greater than Program Amount)
```

(meaning: (=> (> CWE PROG_AMT) (- CWE PROG_AMT))))⁴⁹
)

Suppose the FOCUS query generator is requested to produce FOCUS code again. It can extract information from the blackboard about Overrun Cost as:

13. Sorting data items:

((Overrun Cost)

(meaning: (=> (> CWE PROG_AMT) (- CWE PROG_AMT))))

Following the logic of processing as described in the above 8 rules, the query generator will have to execute rule 4, which is repeated here:

Rule 4.

If <term> is-a-member-of Sorting-Data-Item

Then <field> is-derived-by-finding-FOCUS-field-from <term>

<field> is-a-value-of FOCUS-Sorting-Field

The Lisp function which is associated with the derivational object is-derived-by-finding-FOCUS-field-from discovers that <term>, in this case Overrun Cost, is not a field of the CAPCES database. It means that a DEFINE command statement has to be made for creating a temporary data field Overrun_Cost. Taking statement 13 as an input of another set of rules for producing DEFINE command statements, Expert-MCA can produce a DEFINE command statement as follows:

```
DEFINE FILE PMMFILE
OVERRUN_COST = IF (CWE GT PROG_AMT) THEN (CWE - PROG_AMT) ;
END
```

As a result, the complete set of FOCUS code for the query "Show the projects in fy 90 by overrun cost" looks like this:

```
DEFINE FILE PMMFILE
OVERRUN_COST = IF (CWE GT PROG_AMT) THEN (CWE - PROG_AMT) ;
END
TABLE FILE PMMFILE
PRINT CWE AND PROG_AMT
BY OVERRUN_COST50
IF FY IS 90
```

⁴⁹The idea of translation for the meaning slot from the definition slot has not been fully implemented in Expert-MCA currently. The content in the meaning slot is created and used internally by the system when it processes a query. This internal expression (=> (> CWE PROG_AMT) (- CWE PROG_AMT)) is interpreted as IF (CWE > PROG_AMT) THEN (CWE - PROG_AMT), Or in generality, (=> (expression-1) (expression-2)) means that IF Expression-1 Then Expression-2, where both Expression-1 and Expression-2 are a well-formed-formula, with logical operators connected its neighboring operands other well-formed-formula.

⁵⁰This is defined as a temporary field in the DEFINE part at the beginning.

END

Note that FOCUS is able to deal with multiple temporary data fields up to 256 items, in conjunction with other conditions. As described in Section 4.2, more complex concepts or data items can be defined as procedural terms, which generally corresponds to temporary data fields that require more complex steps for producing DEFINE command statements as pre-processor of TABLE command statements, or that needs post-processing after data retrieval from the database.

7.2 Knowledge Acquisition and Sharing in the System

Knowledge acquisition into a knowledge-based expert system is always in great demand. System designers cannot predict all the needs users may have in the future. Nor can they pre-define how users view the domain world and formulate the domain world in their problem solving process.

The most fundamental and handy tools that humans communicate with each other is natural language. As a result, Expert-MCA is designed to acquire knowledge mostly via adding new terms into its dictionaries. Such terms are English words or English phrases, with definitions describing concepts, objects, or chunks of knowledge. In other words, terms are the fundamental unit defined to the Expert-MCA and used to compose queries. They are also the knowledge unit employed in Expert-MCA to process knowledge for finding information and solving problems.

7.2.1 Knowledge Acquisition in Expert-MCA

The module used to acquire knowledge in Expert-MCA is the Teaching session. As defined in Section 4.2, terms are classified into six types. To start defining a new term or modify definition of an existing one, the user is asked to select a term type for the term. Next, the user has to type the term, which can be a single word or multiple words. If the term entered has been defined before, the system will allow the user to make a choice among the following: to define a new one, to edit an existing one, or to quit.

Based on the type of the term being defined or edited, Expert-MCA provides the user with an appropriate template, which is arranged like a table with row names and blank spaces to be filled out. Each of the rows corresponds to an attribute of the term object. For

example, a term may have the following attributes (or slots): **Part-of-speech, Definition, Context, and Comment**⁵¹.

A Help-Define mechanism has been implemented to direct the user to fill out the slots of a term. The idea employed in the mechanism is that the syntax of slot definition can be used to guide how to construct valid statements for the slot. For example, if we define the syntax of the Definition slot for a **Word/Phrase in Backus-Naur Form** as follows:

```
<Definition-slot-content-syntax;>
 ::= <flv> | <flf> | <faf> | <fav> | <fav-if> | <faf-if> | KEY-IN
<flv> ::= <fieldname> <logical-operand> <value>
<flf> ::= <fieldname> <logical-operand> <fieldname>
<faf> ::= <fieldname> <arith-operand> <fieldname>
<fav> ::= <fieldname> <arith-operand> <value>
<fav-if> ::= <fieldname> <arith-operand> <value> <if-statement>
<faf-if> ::= <fieldname> <arith-operand> <fieldname> <if-statement>
<if-statement> ::= <if-flv> | <if-flf>
<if-flv> ::= IF <flv>
<if-flf> ::= IF <flf>
<logical-operand> ::= GT | GE | LT | LE | EQ | IS | NE
<arith-operand> ::= + | / | * | /
```

where

- <fieldname> is a field of the CAPCES database
or a derived-item (i.e., temporary data field);
- <value> is an alphanumeric or a number;
- KEY-IN means that the user will enter the current word by himself.

Examples of valid statements for the Definition slot are listed as follows:

Types:

Examples for the contents in the Definition Slot:

<flv>	Program Amount gt 1000
<flf>	Program Amount gt Current Working Estimate
<faf>	Program Amount * some-special-code-of-a-field
<fav>	Program Amount * 0.04
<fav-if>	Program Amount * 0.05 IF Fiscal Year gt 90
<fav-if>	Program Amount * 0.05 IF Program Amount gt Appropriated Amount
<faf-if>	Current Working Estimate / Program Amount

⁵¹Detailed specifications for terms are defined in Section 4.2 and Appendix B.

IF Program Amount gt 0
<faf-if> Current Working Estimate - Program Amount
IF Current Working Estimate gt Program Amount

Suppose the user is going to fill out the Definition slot of a Word/Phrase term. Now the cursor is on the text of Definition: in the screen template. The default mode in a process of filling out a slot is manual. If the user is not familiar with the syntax of the slot. Nor is he or she aware of what are the terms that he or she can fill in the slot. By pressing a special key, according to on-line help as shown on the bottom of the screen, the user can switch to a Help-Define mode. In a Help-Define mode, Expert-MCA dynamically provides the user with a list of options which are determined by locating the input status in a syntactic tree, which is built based on the slot syntax, such as the one described above. The syntactic tree for the Definition slot is graphically shown in Figure 7-1.

After the user presses a special key, F6 in this case, a one-line display window on the top portion of the working window prompts the options FIELDNAME and KEY-IN in horizon. If the user selects FIELDNAME, a session of selecting a field that is defined in the dictionary proceeds. Such a field selection process will be detailed in Section 7.2.2. For the time being, assume the user has selected one from the field selection process, and the selected field will be automatically filled into the current slot and echoed as well on the screen as if it were filled by the user manually. Next, the one-line display window shows the options: LOGICAL-OPERAND, ARITH-OPERAND, and KEY-IN. If ARITH-OPERAND is selected, its following options shown on the display window are a list of all the defined arithmetical operands. After an arithmetical operand is selected, the selected operand will be filled into the current slot and the display window shows the new options: FIELDNAME, VALUE, and KEY-IN. As such, the user is guided to construct validate statements step by step.

7.2.2 Knowledge Sharing in Expert-MCA

To share knowledge with others, users have to exchange with each other the dictionaries of term definition. To accumulate knowledge in Expert-MCA somehow means to update dictionaries. These are actually accomplished by reviewing or editing the definition of terms.

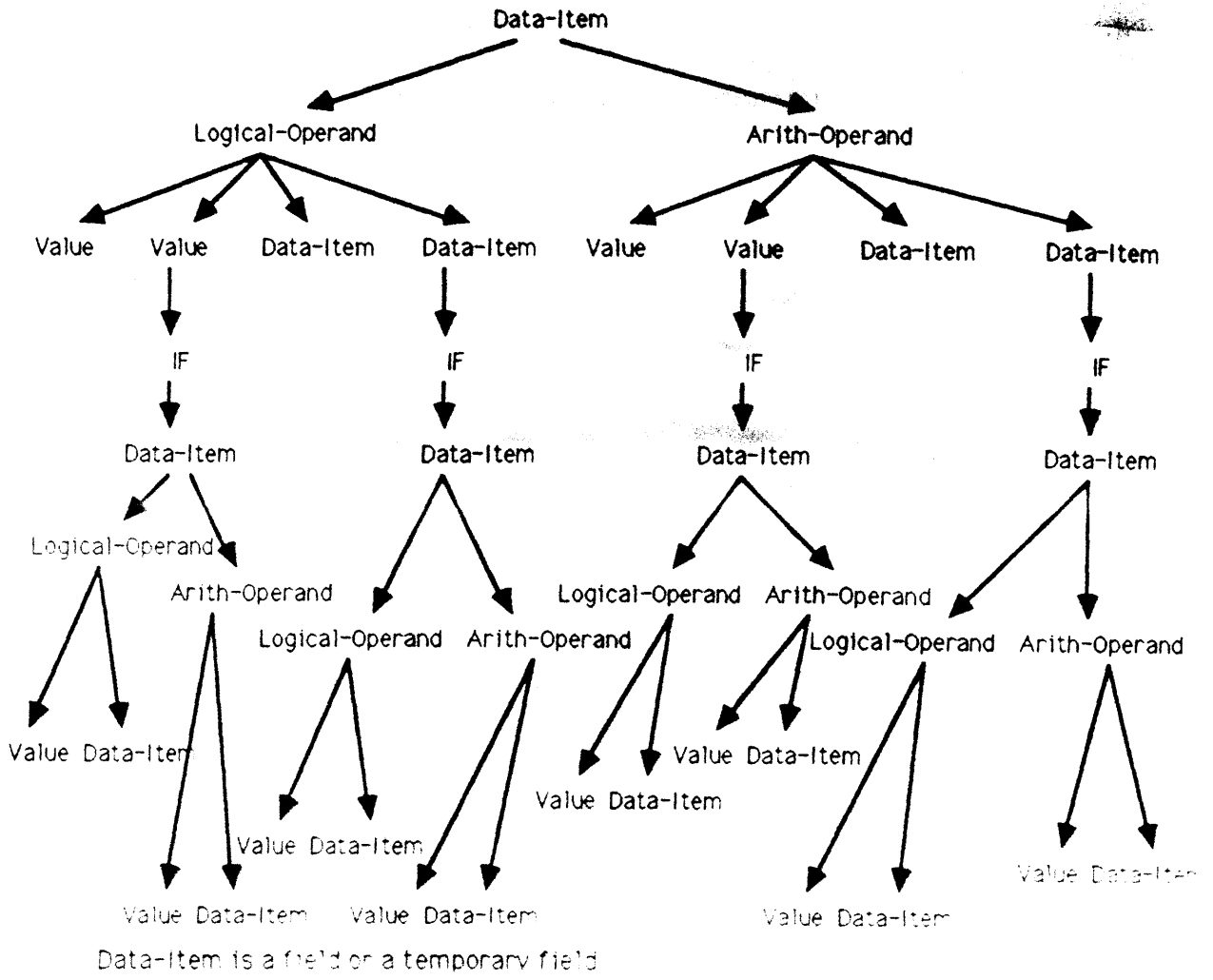


Figure 7-1: A Syntactic Tree for the Definition Slot

It is not easy for a casual user to find a term of interest or to know what terms are available in defining a new term, particularly when the term dictionary is getting larger. An algorithm to aid users in searching for a term has been implemented in Expert-MCA. Expert-MCA establishes a context map by associating all defined terms with a list of contexts (or keywords). To find a target term, the user has to walk through a context map, with each step meaning that a context is collected. Based on the collected contexts so far, Expert-MCA dynamically determines a list of candidate contexts to be selected next.

The algorithm is described as follows:

1. Define Context-Collected as a set consisting of the contexts collected so far.
2. The first few terms whose associated contexts are a super set of Collect-Collected are shown on the screen. Such terms are also called qualified terms at the moment when the contexts in Context-Collected are considered.

3. Collect the terms that are associated with a set of contexts which is a super set of Context-Collected. Define such a set as Term-Context_i for such a term i.
4. Union the contexts that are in Term-Context_i but not in Context-Collected. Define the set of such a union as Candidate-Context. The members of Candidate-Context are the contexts shown on the screen to be selected by the user.
5. Update the set Context-Collected. If a new context is selected, push this new context into Context-collected. If a QUIT option is selected or a special key for the previous screen is pressed, the last selected context is removed from Context-Collected. If EXIT option is selected, the process of walking through the context map is over and Expert-MCA will switch to a new session of selecting a target term from a list of qualified terms.
6. Go to step 2 for another run of selecting a context.

In other words, the system provides users with a context map, which is used to automatically suggest a next level of available context options/directions after a context is selected. If the the next level of options seems no good, users can go back to the previous one and reselect an option. A set of qualified (candidate) terms according to the contexts selected so far will be prompted to help users decide which way to go next. This process goes on until the EXIT option is selected. Now the user can proceed to select the desired term form a menu with qualified candidate terms.

Chapter 8

Implementation Details

Expert-MCA operates under the GCLisp Developer, a product of Gold Hill Company, of Cambridge, MA, on an IBM PC/AT level personal computer. It requires at least three megabytes of RAM. It has been written entirely in GCLisp and runs with mostly compiled code. Part of the communication package called COMMLisp, a product of Brodies Associates Co. of Boston, MA, is modified and incorporated into Expert-MCA. To run queries against the databases stored in PCs, installation of a package called PC/FOCUS is needed.

8.1 Principles and Modules in the Programming Design

8.1.1 Programming Design Principles

Object-oriented programming languages have proven to be beneficial in such aspects as data abstraction, modularity of code, and ease of maintenance. With this in mind, Expert-MCA has been designed with an object-oriented programming style. In addition to the characteristics such as modularity, data abstraction, Expert-MCA is aimed at providing programmers with ease of code maintenance.

Expert-MCA is a package designed to fit the need of frequently interfacing with end users. Therefore, the overall programming flow of Expert-MCA is derived from a user's point of view, and is established around the concept of interface tasks in which Expert-MCA and users can communicate with each other in a friendly fashion.

From a user's point of view, what he or she needs to do or can provide to Expert-MCA can be identified as the following action types:

1. a decision making via a selection from a menu which consists of several options;
2. a text input for simple communication;
3. a series of selection input for searching desired information or making a decision;
4. a series of text input which all together contributes to form a complete set of information to Expert-MCA;

5. a review of messages provided by Expert-MCA.

Most users do not concern how Expert-MCA arrives at its answers. Instead, they tend to make easy decisions to instruct computers what to do next, and thus leave dirty works to computers until it becomes a need to tell computers how to do in another situation. Dirty works in the application domain of Expert-MCA include symbolic matching, information searching, selection of next task, and number crunching.

As a result, a task-oriented formalism is used in Expert-MCA to represent and reason the communication process until an output is worked out. The concept of objects and user action types in man-machine interactions induces the need of task dispatcher for various interface tasks. For each interface task, the task dispatcher decides what and when dirty works should be done internally by computers and leaves decision making via interface tasks to users.

8.1.2 What an Interface Task Does

An interface task is processed via display windows in which Expert-MCA is able to interactively communicate with users. At each interface task, display windows convey useful information to help users make a good decision. Typically, an interface task enables users to do one of the following main tasks:

1. select an option from a candidate list which is prompted vertically;
2. select an option from a candidate list which is prompted horizontally;
3. enter strings in a small display window which performs as a text editor;
4. enter strings in several small windows sequentially; (although each acting as a text editor, together they are associated with a title. This is useful when we are editing content of slots of a frame.)
5. review text or message.

8.1.3 How Interface Tasks Flow from One to Another

After checking for a username and password, Expert-MCA initiates the main menu (or Top-Task) by calling the Lisp function Task-Dispatcher with the argument Top-Task. Depending on the execution result, Task-Dispatcher decides a task which is to be executed next, and then triggers the next task by calling the function Task-Dispatcher with the new selected next task as its input argument. As such, via calling the function Task-Dispatcher one after another, Expert-MCA is able to gain such benefits as: ease of debugging,

independence of task, clarity of task flow, modulation of data flow, and maintenance of a minimum Lisp calling stack group.

Each time the function Task-Dispatcher basically goes through the following steps:

1. Retrieve the definition of the current task (i.e, given the input argument as a task name);
2. Execute a pre-processor if necessary;
3. Execute the main task;
4. Check the output of main task and decide what to do next;
5. Execute a post-processor if the output is not an interrupt;
6. Decide a next task⁵²;
7. Clean the task stack to maintain a minimum stack group of Lisp function calling;
8. Call the next task.

The detailed algorithm for Task-Dispatcher can be referred to the Lisp code in the file c:\mca-task\basic\tsk-main.lsp. The person who is to add or modify Expert-MCA modules is highly encouraged to fully understand the function before he or she starts programming. Note that the definition (or the slot contents) of a task can be updated dynamically as Expert-MCA proceeds in a session.

8.1.4 How to Define an Interface Task

In terms of AI's terminology, an interface task is a frame associated with several slots, each containing a set of information with various formats, such as string, symbol, list, number, executable code, and combinations of the above. The definition of tasks is stored in *.TSK files. For example, the task Top-Task (main menu) is defined in the file c:\mca-task\basic\basic.tsk, and the task QUERY is defined in the file c:\mca-task\query\query.tsk.

The following shows an example of a task definition, followed by a detailed description of how to define contents for slots in an interface task.

Suppose we want to define the main menu (or top menu) in Expert-MCA. The format of a task definition is like a table. Therefore to define a task is just like to fill out a table,

⁵²When the output of main task is an interrupt, Task-Dispatcher will decide a next task based on the type of the interrupt.

with each row being led by a slot name which is followed by its value. Part of the definition for task **Top-Task** is shown as follows:

Name:	Top-Task
Title-Text:	("Please Choose One")
Window-Type:	single-selection-menu
Option:	("Overview" "Query" "Report" "Download" "Teach" "Profile" "Utility" "Logout")
Next-Task:	(overview query report download teach update-user-profile utility exit-to-Lisp)
Brief-Help:	See Below
Detailed-Help:	See Below

Note: This is the Main Menu of Expert-MCA.

The slot **Name:** is used to store the task name being represented. In this case, the value is **Top-Task**. The slot **Title-Text:** is used to indicate the text which is shown on the screen as a title for the menu. The slot **Window-Type:** is used to specify what kind of window type the menu is going to use. In this case, **single-selection-menu** stands for a menu that is used to select a single choice. The slot **Option:** is used to store the options that the menu provides to users. The slot **After-Action:** is used to store actions which needs to be executed after a choice is made from the menu. If it is empty, then just go to the next task. The slot **Next-Task:** is used to store the task names that will be executed after the actions stored in the slot **After-Action:** are done. Note that the sequence of the task names in **Next-Task:** should be parallel to that of the options listed in the slot **options:**. For example, if the second option is selected, then the second next-task, in this case **QUERY**, is the next task to be executed.

The slot **Brief-Help:** is used to store a brief help message that is to be prompted on the screen as the cursor moves around on the menu. The content stored in the **brief-help:** slot for **Top-Task** (or main menu) looks like this:

```
("Overview About Expert-MCA: what it can do for you.")  
("Execute a Query Session for Retrieving Information.")  
("Execute or Review/Update Standard Reports.")  
("Download CAPCES Data to PC As Topical Databases.")  
("Teach or Edit the Terms Used in Expert-MCA.")  
("Modify User Profile and Default Information for Data Retrieval.")  
("Utilities in Expert-MCA Environment.")
```

"NOT implemented yet.")
("Quit from Expert-MCA and Go back to Lisp Top-Level."
"Type (MCA) at Lisp Top-Level to start another session."
"Hit F1 to DOS temporarily, and type EXIT back to Expert-MCA."))

The slot Detailed-Help: is used to store a detailed help message that is to be prompted on the screen when users need more detailed help. The content stored in the detailed-help: slot for Top-Task (or main menu) looks like this:

("General Overview about Expert-MCA:"
"Line 1 is to be filled with the description about Expert-MCA Overview")
("General messages about Queries:"
"Line 1 is to be filled with the description about Queries.")
("General messages about Standard Reports:"
"Line 1 is to be filled with the description about Standard Reports.")
("General messages about Downloading:"
"Line 1 is to be filled with the description about Downloading.")
("General messages about Teaching/Editing Terms:"
"Line 1 is to be filled with the description about Teaching/Editing.")
("General messages about User Profile:"
"Line 1 is to be filled with the description about User Profile.")
("General messages about System's Utilities:"
"Line 1 is to be filled with the description about System's Utilities.")
("Detailed message about what you can do after logging out."))

What follows gives a more detailed specification of defining an interface task. To describe the specification we will apply the terms such as Symbol, String-List, Symbol-List, and Functional-Expr. which are defined as follows:

1. A Symbol is composed of alphanumerical characters, such as English letters and numbers.
2. A String-List is defined with the format: ("line 1" "line 2" ..) Usually, each string element in the String-list conveys the text that is to be printed in a line on the screen. If the input is a single string, it will be converted to a String-List. For example, If input to a slot is "only one line strg", the system will check and convert it into ("only one line strg").
3. A Symbol-List is defined with the format: (symbol1 symbol2 ..). If a Symbol-List consists of a single Symbol, it can be entered with the element without parentheses.
4. A Functional-Expr is an expression consisting of a Lisp function associated with necessary arguments.
5. Each element of String-List or Symbol-List can be a Functional-Expr.

The slot names which are used in Expert-MCA to define an interface task include: Name:, Title-Text:, Window-Type:, Option:, Window-Size, Message:, Brief-Help:, Detailed-

Help:, Before-Action:, After-Action:, Next-Task:, Loop-Over:, Input:, Output:, and Needed-Global-Variables:. The following describes syntax and use of each slot.

1. **Name:** a unique Symbol, must be different from other task symbols. If the slot title **Name:** is not provided in the definition list, then the first symbol right after **DefTask** must be the task name itself (i.e., it is the content for the slot **Name:**). The order of the other slots in a task definition is not significant as long as each slot title is followed by its content. Default content of each slot is set to **NIL** if not provided.
2. **Title-Text:** a String-List used to print as a title of the interface task . The window space for printing the title-text is defined by the variables such as ***menu1-head-window-size***, ***text-input-head-size***. Currently, the size is 3 x 80 in characters.
3. **Window-Type:** a Symbol which defines the working type in the major display window. It must be one of the following types: **single-selection-menu** (for choosing one out of several candidates), **line-text-input** (for an input with a single line), **text-input** (for text input/editor with many lines), **frame-slot-editor** (for defining/editing an object with several slots), and **message-lookup** (for prompting messages).
4. **Option:** a String-List which defines the content of a selection menu. The number of options in a menu is not limited. The text of each option is stored as a string element in the String-List.
5. **Window-Size:** a value list which specifies a window's left-top position & size. This slot is needed only when **TEXT-INPUT** is selected in the **Window-Type:** slot. Content of the slot is used to specify the dimension of the display window for editing text. The syntax for the slot content is (left top width height), where left < 76, top < 10, width < 76, height < 12 in characters. Top stands for a relative margin in lines below the top of ***text-input-window-size***, which holds the size (0 5 80 12) measured in a global screen scale (see **tsk-vari.lsp** file).
6. **Message:** a String-List, or a Function-Expr that will be evaluated when the slot content is needed. The slot content represents for the text to be printed as a message when the **message-lookup** window type is selected.
7. **Brief-Help:** a String-List, or a Function-Expr that will be evaluated when the slot content is needed. The content will be prompted in a **BRIEF-HELP-WINDOW** as a brief message to help explain the situation where the cursor is on.
8. **Detailed-Help:** a String-List, or a Function-Expr that will be evaluated when the slot content is needed. This slot provides a detailed message under the situation where the cursor is on and when users need more information for making a decision.
9. **Next-Task:** a Symbol-List, a Symbol, or Nil. It defines candidate tasks to be run after the current task. The tasks included in the Symbol-List must be arranged in an order corresponding to the options in **Options:** slot. The content in the slot can be a Symbol indicating the next task to run. Also we can leave it

unspecified at the moment of defining the task and later dynamically assign one to it at the time of executing procedures which are specified in the **After-Action:** slot or elsewhere. A **next-task** can be dynamically decided by the procedures defined in the **After-Action:** slot.

10. **Before-Action:** a Symbol or a Symbol-List, the Symbol or each element of the Symbol-List serves as a dynamic procedure to update information into slots of the current interface task. The slot may contain several symbols, each standing for a Lisp function and being executed sequentially. Together, the Lisp functions act as a pre-processor before a main job is executed. Usually a pre-processor is used to modify the initial input to the current task or to update contents of other slots before its main task starts. The updated input (if any) then will be sent to main action for further executions. However, the **Input:** slot still holds the original input. By doing so, if the interface task is triggered by a later task, it will perform exactly the same as it does now.
11. **After-Action:** a Symbol or a Symbol-List, the Symbol or each element of the Symbol-List serves as a dynamic procedure to update information into slots of the current interface task. This action may modify output of the current task or direct the system how to select an appropriate one among those candidate tasks stored in the **Next-Task:** slot. If output of the current interface task is not a number which stands for a selection from an option menu, the output will be sent to the **Input:** slot of the next interface task. Users do not need to specify arguments for the functions specified in **Before-Action:** and **After-Action:** slots. Instead, the function **Task-Dispatcher** will find one, since the only argument for the functions stored in **Before-Action:** slot and **After-Action:** slot is the interface task name itself (see **Input:** and **Output:** slots below).
12. **Loop-Over:** This slot is updated by other actions. Based on the content stored in the task, **Task-Dispatcher** can repeat a series of interface tasks until the slot is consumed completely.
13. **Input:** and **Output:** The two slots are used by the function **Task-Dispatcher** to pass arguments between any two interface tasks or the functions specified in **Before-Action:** and **After-Action:** slots. We do not fill out these two slots in defining a new interface task. In cases that some parameters are needed globally, we can define global variables or parameters. As a good convention, we may name global variables having the character "*" both at the beginning and the end of the variables.
14. **Needed-Global-Variables:** This slot stores global variables used in the current interface task. The slot is optional. It is used to indicate the stored global variables may be updated inside this slot. It is a good convention for program maintenance and debugging.

Note that users are not supposed to use quote (i.e., the mark "'") in front of slot content, since the Lisp function **DefTask** knows how to add quotes in front of slot contents. Users have to make sure that every slot title ends with ":". Definitions of interface tasks are stored in files with a file extension ".TSK". The functions referred to in **Before-Action:** and

After-Action: slots for an interface task are stored in a file that has a file extension ".LSP," preceding with the same file path as the file storing the interface task.

8.2 Flow of Modules in Expert-MCA

The entire display screen of PC is divided into several rectangular areas for different types of messages or functions. The top one, called top-line window, shows the name of current interface task, date, and time. The one right below is the working area that users respond to Expert-MCA. This working area, called major window, may occupy a space with a height from 15 to 22 lines. The next one is a one-line area indicating the mark "Construction Engineering Research Laboratory of the USA Army." Below this mark is a five-line window called brief-help window, which prompts brief help message corresponding to the situation where the cursor is on. The bottom window, a two-line display window, shows on-line help about the function of special keys, such as F1, F2, etc.

All user interface screens (or interface tasks) are defined in files *.tsk, which can be modified as needed. For example, the file Basic.tsk defines some basic templates with slot titles and associated tokens. Most tokens are used to define syntactic formats for slot contents. Such templates can be used in defining new terms or other tasks (such as update-report, update-download, and update-profile for tasks in sessions REPORT, DOWNLOAD, and PROFILE, respectively). The interface tasks defined in Expert-MCA are listed in Appendix D, each being attached with its major definition content. A list of the files being used in Expert-MCA is also given in Appendix E.

Major functions within the system are querying the system and its database (i.e, Query Session), knowledge acquisition (i.e, Teaching Session), downloading a subset of the mainframe database to the PC (Download Session), and providing user profile data and domain knowledge (Profile Session). The system also can provide access to preprogrammed mainframe reports (Standard Report Session).

The Profile Session has two branching sessions: User Profile Session and Domain Session. The user can specify information about his (or her) job background (such as agency, the projects in question), communication parameters (such as a username for entering the mainframe, baud, port, telephone number, etc.) and some default data items and screening conditions that are needed in answering a query. In the Domain Profile Session, the user can specify propositions that are related to the domain and language use in queries,

names of rule bases for inferencing information to refine queries, names of objects that are defined by the system and the user.

The first interface task is Top-Task, which serves as the main menu of Expert-MCA. Currently the available interface tasks from the Top-Task are OVERVIEW, QUERY, REPORT, DOWNLOAD, TEACH, PROFILE, and LOGOUT. At any point in running Expert-MCA, users can go back to the main menu or the previous screen (or task) by hitting a special key as indicated on the bottom window.

Figures 8-1, 8-2 show the flow of the interface tasks used in Expert-MCA.

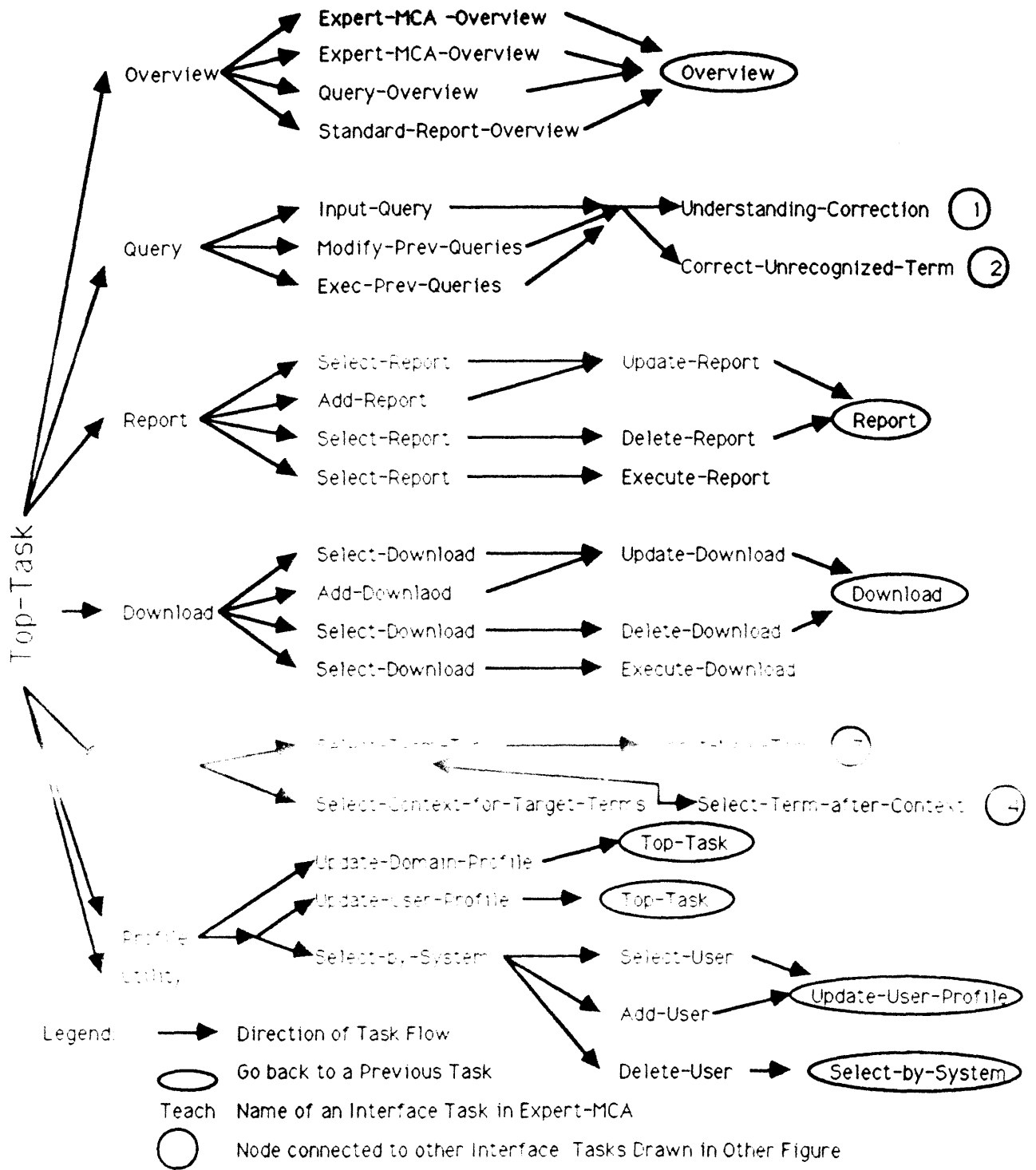


Figure 8-1: Interface Task Flow in Expert-MCA, Part 1

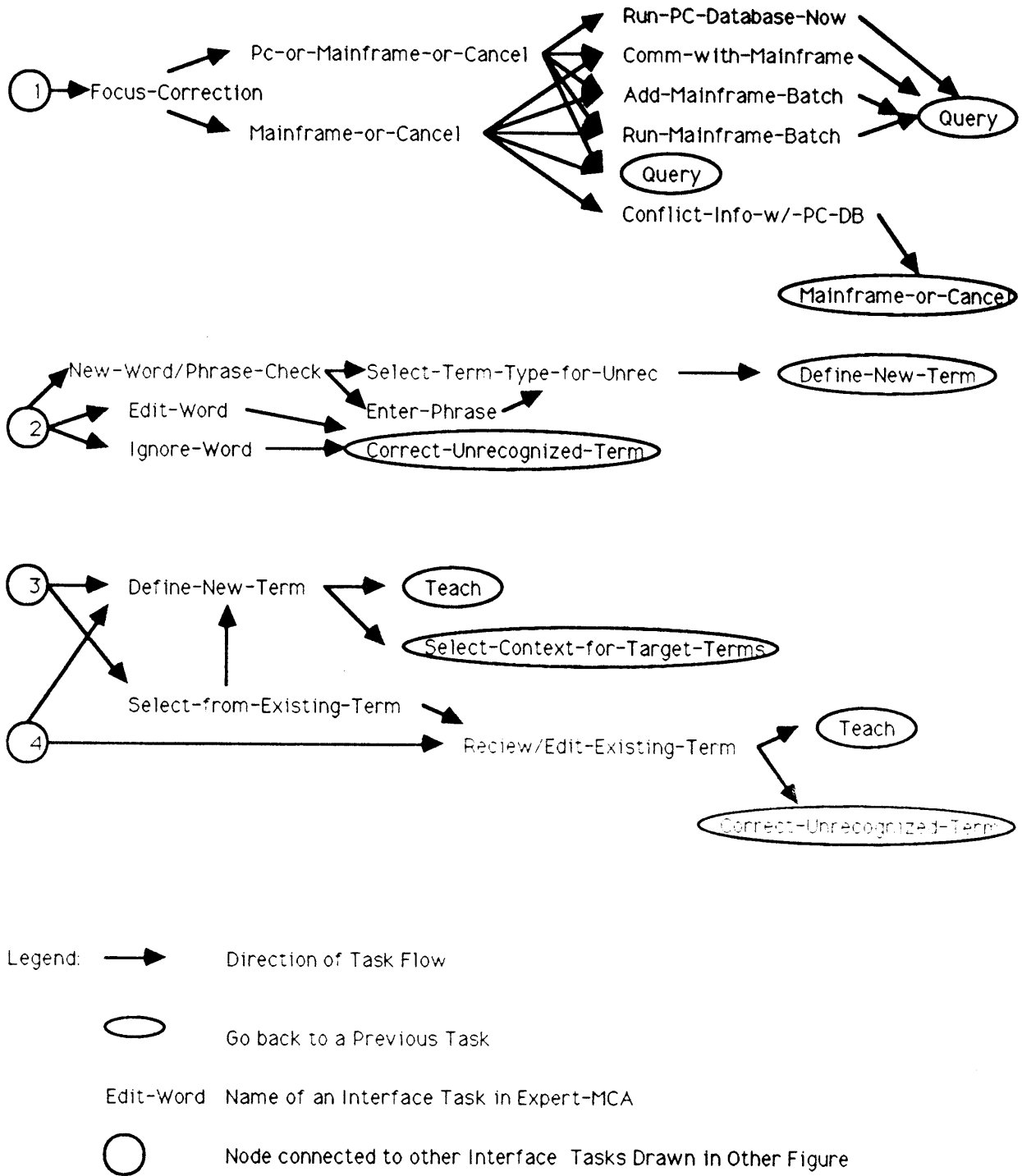


Figure 8-2: Interface Task Flow in Expert-MCA, Part 2

Chapter 9

Conclusion and Recommendation for Further Study

This chapter summarizes this thesis work and recommends future directions in the line of knowledge processing for the construction domain.

9.1 Conclusion

With such deleterious characteristics as fragmentation, transient nature, non-repetitive process, ineffectiveness of learning, and inefficiency of communication, the construction industry today has been further compounded by the ever increasing complexity of construction projects that require more specialized equipments and skills, that suffer stricter regulations, and that are encompassed by higher pressures for shorter delivery duration. This strongly suggests that we may need more powerful tools, such as knowledge processing systems, to help manage the problems in construction management.

Just as do database management systems facilitate users the use of data at present, so can knowledge processing systems help the practitioners improve their learning effectiveness and communication efficiency. Specifically, knowledge processing systems can be used to organize the domain's knowledge structure, collect and maintain knowledge systematically, and infer new facts automatically. Such systems are deliberately integrated by incorporating various knowledge sources and adding newly available information processing technologies to better make use of the strengths from individual ones. They are coordinated with better information flow procedures in an organization; allowed to have a broader use of information; dynamically customized to meet user's needs; able to automatically reason about new facts.

Knowledge processing typically consists of the following steps: knowledge formalization, knowledge representation, and knowledge reasoning. AI researchers have been working on the subjects of knowledge representation and knowledge reasoning, while leaving knowledge formalization to researchers in application domains.

This study has been working on the knowledge based query system Expert-MCA to demonstrate how a knowledge processing system can be designed and implemented as a

system that can better utilize domain knowledge and information by integrating today's information processing technologies, mainly drawn from AI and MIS areas. It has been deeply motivated by how we can improve communication efficiency and learning effectiveness in the domain of construction engineering and management. The construction management database, CAPCES, which supports various staff of the US army for making decision about their construction project programs, serves in Expert-MCA as the primary source of information requested by the staff via near-natural language queries.

Expert-MCA has shown how a knowledge processing system can alleviate the burden of learning details of a database and a database management system in order to get some useful information from an existing database. It also shows the coupling of data stored in a database with heuristic knowledge in order to increase the value of both. Although the knowledge based query system only deals with a narrow domain of the U.S. Army construction program, it illustrates improvements in the communication between users and a database that contains broad classes of data from multiple components of an organization. At the same time, the coupling of an expert system to the database management is shown to help the database management system user to better understand the data and use them to provide more intelligent decision making. On the other hand, the teaching facilities in Expert-MCA offer mechanisms for accumulating knowledge in the military construction domain, and hence may provide its casual users with the chances to share the knowledge extracted from more experienced colleagues.

9.2 Discussion

The use of near-natural language queries, when combined with user's maintenance of terms, in Expert-MCA provides users with adequate power and ease in expressing requests for database information. All database fields exist in the lexicon, so do common synonyms for field names and terms for field values and value ranges. The user may then compose a query in terms that are English words or terminologies used in the application domain. By providing their own definition of terms, users can express their requests as they wish. Rules for query construction (which need not be rigidly followed) follow at worst a simple syntax, starting with requested data items, followed by screening conditions for selecting desired data instances, and then followed by sorting data items if necessary. By providing the user near-natural language query tools, he or she can be relieve of a programming burden, either

for himself or having to deal with programmers to translate English terms into corresponding field names and values for a target database. This is particularly true in the case of large, multi-user databases such as the CAPCES database, which contains over 500 field names, most with names not close to natural language expressions and with tables for meaning of their values.

Online help, clear messages, and guidance throughout the process make the use of Expert-MCA easy for the casual user. If a user gets lost, he or she may interrupt processing and go back to top level to start again and rephrase the query or use keyword or topic searching to look at definitions of the terms which refer to his or her need. On the other hand, a more knowledgeable user may define new terms, procedures and rule bases, so that the potential users may range from a sophisticated programmer to casual clerks. The system is designed to fit the needs of users with different levels of knowledge backgrounds on the CAPCES database and MCA construction program.

Defining terms and the procedures they call, such as the procedures illustrated in Section 6.5, provide the user with the ability to extract and manipulate the data in complex ways. While the capabilities (inferencing, for example) go beyond those available in query language alone, including FOCUS, their use does require programming skills. It is believed that Expert-MCA provides a language simpler than FOCUS and yet more powerful. It is simpler in that it allows the use of terms instead of database fields and values, does not necessitate database structure knowledge, and coordinate database retrieval. It is also powerful in that it can direct reasoning about retrieved data and contexts involved. Its programming style facilitates such benefits as data abstraction, code modulization, and code reuse. It is a language with many characteristics of FOCUS and procedural languages such as FORTRAN.

The approach used in the system is not limited to retrieval of data from a single file or table. The use of an internal representation makes this possible. User's input queries are superficial representation because they may imply more meaning than what is conveyed literally by each of the English terms. An internal representation is formed to represent the meaning that includes the literal one transformed from the input queries and the embedded one derived by Expert-MCA. The use of an internal representation is two-fold. It facilitates the reasoning process internally in the reasoner module. It also provides a transition framework for further transformation to a target formal query language such as FOCUS or SQL. Although currently Expert-MCA is limited in the generation only to the target query

language FOCUS, its query language generator can be replaced with a new one aiming at producing another target query language.

Most DBMS focus on how the data can be effectively and efficiently retrieved. This system, however, tries to focus on how information can be extracted from the data and how knowledge can be accumulated for this extraction process. In construction management, having handy report generators for data retrieval is only a tool for the first step in good decision making. The more important and challenging issues are how we can derive and accumulate our knowledge from the data collected in existing databases, how we can transfer such knowledge from one to another. After all, the difference between a sophisticated construction manager and a casual one is not because the former does know more about how to retrieve data or generate reports. Rather, the former does know more about the interpretation of data items, the possible patterns among data items, and the implication of some data instances to other data instances.

This study has experienced many difficulties in shaping its research framework. It does not attempt to develop a generalized parser for processing a large set of natural language, nor does it deal with the data processing other than data retrieval from a single database. Although Expert-MCA has not been tested in the field, two issues important to its use are: how the user is capable of composing queries and of defining new terms. Since Expert-MCA may fail to understand user's queries, the user may be frustrated in a sense that he or she does not know what queries are acceptable and what else are not. On the other hand, the user may be puzzled to act in defining a term, mostly with the unawareness of what terms or the syntax he or she can use. To reduce the possible perplexity, the system has provided a Help-Define module to guide, with a selection menu, the user to define a term correctly.

As compared with other natural language front-end systems used to interface with existing databases, mostly for data retrieval only, Expert-MCA has shown many distinct features in parsing. It is designed to retrieve data by using information about, in addition to syntax and semantics, the contexts involved, user profile, and domain pragmatic practices. Although it can be weak in analyzing compound sentences syntactically, its mechanisms for users to define knowledge for forming split terms, resolving ambiguities, specifying meaning of terms or data items, and further processing on retrieved data (in coupled with rule based inferencing or procedural processing) may well go beyond conventional ones, such as the INTELLECT system.

Table 9-I: Comparing Expert-MCA with Related Systems

Systems Features	Expert-MCA	Q & A	Intellect	LN I	Knowledge Craft
Use of Near Natural language	Y	Y	Y	Y	Y
Operation On PC	Y	Y	N	N	N
Help Define New Terms	Y	N			
Use User Defined Knowledge	Y	N			
Rule Based Inference	Y	N			
Procedural Processing	Y	N	N	N	N
Access to PC Database	Y	Y	N	N	N
Access to Mainframe Database	Y		Y	Y	Y
Access to Damain Knowledge	Y	N	N	N	N
Understanding Correction	Y	N	N		
Query Language Code Correction	Y	N	N	N	N
Explanation of Reasoning Process	N			Y	
Data Model/Query Language	Hierarchical FOCUS	N	Relational	Relational	
Expert-MCA does not focus on			ATN	Case Frame Deductive Quantfier	Case Frame

Note: Y: Yes, N: No, Blank: Unknown

Having been designed as an integrated system, Expert-MCA has many features that related systems are weak or not provided. A comparison of Expert-MCA with related systems about some important features is given in Table 9-I.

The syntax used in the system for defining procedures and rule bases is still incomplete in a sense that one cannot foresee all the information manipulations required in solving user's problems. Although Expert-MCA has simple syntax for composing simple statements (or propositions), the transformation from such statements to their underlying internal representation is non-trivial. How the information is aggregated or disaggregated for further processing is also not simple in executing complex procedures.

9.3 Contributions

Expert-MCA is a knowledge based query system, notably with an integration of user defined knowledge in database query processing. The contributions of this research, via problem identification and the development of Expert-MCA, can be listed as follows.

1. Integration of information technologies for information retrieval

Expert-MCA has integrated natural language processing with rule based inferencing and procedural processing to retrieve information from a database or other information sources.

2. Developing an ease tool for access to database

Users can use near natural language to query a very large database via an automatic communication mechanism.

3. Providing a medium for understanding meaning of data

Meaning of data and patterns among data items can be captured in definition of terms. Various types of terms are classified in Expert-MCA to help understand their meaning via a friendly user interface.

4. Establishing an environment to define knowledge

Users can define knowledge about the application domain, mostly in a form of rules and procedures, and their language use. User defined knowledge can also be specified in frames of user profile and domain profile.

5. Exercising of user defined knowledge

User defined knowledge is used for translation from English terms to database fields or

ranges of values, access to database or other information sources, pre-access and post-access processing, or definition of user's characteristics to establish necessary context.

Although the application domain of Expert-MCA is the Military construction programs, its underlying ideas and approaches can be used in other areas.

9.4 Recommendation for Further Study

It is believed that knowledge processing systems, mostly integrating the technologies from AI and MIS research results, will be getting more recognition in their use in the construction domain. This trend has been exemplified by many works, such as in [23, 36, 49, 54]. However, it seems that most researchers have been working on various subjects without consulting with each other about fundamental design philosophy and implementation tools. Rather they tend to embody the design philosophy they see the most important, and select the tools that are available to them. While these works continue to evolve independently, it would be better to have an understanding about the directions both in AI application research and MIS research.

Natural language is the most natural and fundamental tool for humans to communicate with each other. Although graphics, tables, or mathematical equations can be a very effective and efficient representation system for a specific domain, the meaning embedded in them can all be described, or substituted, by using natural language. However, the reverse is not always feasible in many domains.

In highly complex, loosely coupled domains, such as the construction engineering and management, natural language tends to be the ultimate tool with which the users can request knowledge processing systems to solve their problems. The obvious benefits that can be gained by using natural language as a communication tool between humans and computers need not any elaboration here.

Currently, the ambiguity-prone characteristics inherent in natural language have greatly impeded the research progress in natural language processing. A challenging research issue is how one can make computers understand natural language expressions effectively in a narrow domain, if he is not able to generally attack all issues on natural language processing as a whole. We need some kind of internal representation systems which allow computers to reason and derive correct meanings that are explicitly conveyed and perhaps implicitly implied by clues in natural language expressions, in a way similar to humans reason from the expressions.

The next issue is how we can construct more expressive representation systems for the construction domain, which is involved with a complex knowledge structure. Object-oriented formalism has been a pervasive concept in building a knowledge processing system, for objects are an intuitive representation in the real world. Such a formalism has been further strengthened by the development of object-oriented languages. However, a lot of relevant issues still remain unsolved. For example, How can we store various types of knowledge and information expressively enough? How can we allow users to, both explicitly and implicitly, relate one object to another by what kind of relationships in order to model the real world⁵³? How can we derive or reason about new information along the relationships between objects or throughout whatever available methods and paths⁵⁴? How can we retrieve information efficiently? Such issues will be further complicated if one considers the integration or coordination of different systems, such as knowledge-based expert systems and database management systems. While today's database management systems have encountered many unsolved issues as illustrated in the so-called composite information systems [56, 85], the integration of various knowledge sources resided in different knowledge-based expert systems may be required and many problems resulted from this integration remain open for further research.

Another issue is what reasoning mechanisms underlying a representation system should be provided in order to solve problems in a domain involved with richer contexts. For example, considering how a person can understand English sentences quite effectively and efficiently, one may realize that to solve ambiguities in sentences needs a lot of contextual information for reasoning about correct results. As described in Section 4.4, context driven reasoning is only a start to enhance computer's reasoning capability that goes beyond the ones we often see today.

⁵³For example, how a concept design is related to a final design? How the concept design is related to the total construction cost and to the total duration of its delivery?

⁵⁴For example, how drawings, specifications, documentations are related to each other? How can one be derived or automated from another?

Appendix A

Problems in Natural Language and Processing Issues

A.1 Problems Inherent in Natural Language

Natural language is an efficient system for transmitting ideas among human beings. From a sociocultural point of view, human beings cooperate with each other to gain all kinds of benefits in their daily life. Such cooperation often requires communication via a medium with desired characteristics such as being easy to learn and easy to manipulate. As a result of progress in its development, the medium, which we call natural language, becomes flexible and succinct in its use via sounds or symbols. People have learned it since childhood without any difficulty. We not only can easily and flexibly express our ideas in natural language, but also are able to efficiently and effectively figure out meaning for expressions.

However, natural language gains succinctness and flexibility in its use at the cost of proliferating ambiguities in interpretation. Succinctness means that the form for expressing an idea tends to be as succinct as possible in language use. Yet the succinctness often results in words that have multiple uses, words that are used to refer to some other sentence constituents, phrases that are made up of several nouns, or sentences that are expressed in elliptical forms. For example, the word **BOOK** can be used as a noun and a verb. When used as a verb, it can also have different meanings such as to reserve a flight, to charge somebody with a crime, or to go very fast. Therefore use of the word **BOOK** may cause ambiguities in syntactic analysis, because it involves multiple word classes and multiple senses.

Flexibility means an idea can be expressed by different sequences of words, or a sequence of words can be used to represent different ideas. For example, to indicate a state with a name, Massachusetts, we may at least have the following symbolic representations: "the state of Massachusetts", "Massachusetts state", "state Massachusetts", "the Commonwealth of Massachusetts", or "the Bay State." On the other hand, the symbol "dishwashers" can indicate machines that wash dishes, or people who wash dishes.

As can be expected, such flexibility may cause people to create a sentence that can be associated with multiple syntactic structures and multiple interpretations. Being grouped word by word, a sentence has a linear structure in the arrangement of its components. Yet

syntactic relationships among sentence constituents are more like a tree structure. The linear arrangement of words in a sentence obviously makes it difficult to clearly specify word relationships in syntax and in meaning. This difficulty, for example, is reflected in the so-called modifier attachment problems, which means a constituent can be syntactically attached to several other constituents in the same sentence. The prepositional phrase "with a telescope" in the sentence "The silly robot saw a box on a hill with a telescope" can be used to modify any of the words SAW, BOX, and HILL. Contextual information is required to resolve such problems.

Understanding natural language is a cognitive process which requires a variety of knowledge, including knowledge of language, knowledge of the world, and knowledge of situations during a communication process. Knowledge of language primarily consists of the following categories: phonology, morphology, syntax, and semantic [2, 64, 91]. Phonological knowledge concerns how words are realized as sounds, while morphological knowledge concerns how words are formed from morphemes, which are basic meaning units in natural language. How words can be grouped into phrases and how these phrases into sentences are the major components in syntactic knowledge. Semantic knowledge is dealing with meaning of words, phrases, and sentences as a whole. How well one can understand a sentence solely depends on how well he or she is able to make use of the knowledge of language, coupled with the knowledge on the contexts involved and on the domain being discussed in a communication process. Constituents in a sentence and the sentence as a whole can be ambiguous in structure and in meaning, if some knowledge required as described above is not clear.

Ambiguity problems in natural language exist in many aspects. English, for instance, may occur on a lexical level, a syntactic level, or a semantic level. These three levels all have to do with two kinds of subproblems: representation problems and interpretation problems. Namely, a representation (either in terms of symbols or sounds) can have several interpretations, while an interpretation may be associated with multiple representations as well. The examples of "dishwashers" and "state of Massachusetts" above also illustrate the representation problems and the interpretation problems, respectively.

Lexical ambiguities occur when words have multiple word classes or multiple senses. The word BOOK, as mentioned previously, may exhibit such ambiguities. Syntactic ambiguities occur when a sentence has words with lexical ambiguities, or when constituents of a sentence can be structured in multiple ways. Most prepositional phrases can be

syntactically associated with several constituents in the same sentence, as illustrated in the previous example about the prepositional phrase WITH A TELESCOPE.

Semantic ambiguities occur when a sentence has lexical or syntactic ambiguities, or sentence constituents have multiple interpretations. With some nominal phrases, for instance, it is difficult to specify their meaning. The nominal phrase "toy factory" may have two interpretations. One is talking about a factory that produces toys, while the other is about a factory that is played around in by kids. Moreover, a modifier can change ways of interpretation of a nominal phrase. For instance, a "toy elephant" is not an elephant. Rather it is a toy that has a shape similar to an elephant. In addition to such nominal compound nouns, referential problems can also contribute to semantic ambiguities. What constituent a pronoun or a determiner refers to is not always so obvious. Sometimes it is not so straightforward to recover the full equivalents of the sentences which are expressed in terms of elliptical forms or incomplete forms.

A.2 Major Issues in Natural Language Processing

The major modules in the analysis for transforming a sentence into an internal expression are a parser and an interpreter. A parser deals with syntactic analysis, while an interpreter deals with semantic analysis and contextual analysis. The internal expression is also called logical form, which is designed in such a way that the natural language processing system is able to reason or manipulate information internally. Using semantic information such as the meaning of words, selectional restrictions or referential objects, the interpreter will rule out some ambiguities that cannot be resolved by a parser which uses syntactic information only.

Depending on how syntactic knowledge and semantic knowledge are employed, systems built for natural language processing can be categorized as a multiple-phased system, a single-phased system, and a parallel system. In a multiple-phased system, syntactic analysis and semantic analysis are kept separate. They are executed sequentially in different stages. Examples of such system are LUNAR [95] and IRUS [9].

Single-phased system processes input sentences in a unified module. Basically, this unified module have three types of structure. The first one matches input sentences with predefined pattern templates. Each pattern template is associated with a predefined format for interpretation in terms of an internal representation for further processing. Early natural

language processing system, such as ELIZA [89] and STUDENT [11], has good performance in a very small and simple domain by using this template pattern formalism. The second type is called Semantic-Based-Grammar system. In this system words are categorized according to a set of semantic categories such as names of attributes, values of attributes, and question marks or other keywords frequently appear in the sentences used for a domain of interest. Grammatical rules in this system are represented as templates that primarily consist of semantic categories. Each template is also associated with a description of how to interpret the sentence which is matched with the template. The description, of course, is expressed in an internal representation. Note that the grouping of new constituents in this system is not based on syntactic categories (i.e., parts of speech). Instead, it is based on semantic categories. Examples of a Semantic-Grammar-Based system include PLANES [83], LADDER [32], and REL [75]. The third one has a set of uniform grammatical rules for analyzing sentences. Each rule simultaneously contains syntactic and semantic information in a uniform representation. Functional-Lexical-Grammar systems fall into this type [12, 42, 91].

A parallel system has two sets of rules for syntactic knowledge and semantic knowledge, respectively. Each syntactic rule has an associated semantic rule. Each time a syntactic rule is applied, its associated semantic rule is also executed. The purpose of parallel execution of semantic rules is to detect early on constituents that are inappropriate as far as semantic constraints are concerned. If such a constituent is detected, the system will give up trying to combine this constituent with other constituents to build up an even larger constituent.

The immediate constituent approach and functional approach are two major approaches used in syntactic analysis. The researchers that primarily follow Chomsky's view have been designing syntactic analysis based on the notion of the immediate constituent since his milestone work in the book Syntactic Structures in 1957. Chomsky's emphasis on syntactic aspects of language has brought tremendous influence on natural language research work both in linguistics and computer science ever since. For example, transformational grammar has led to the development of parsing tools such as transition network, recursive transition network (RTN), and augmented transition network (ATN) for the complexity of English. In this approach terminal constituents (i.e., words) are grouped into nonterminal constituents and then these nonterminal constituents are further grouped into even larger ones. If a sentence can be successfully generated or consumed in terms of constituents, then this sentence is recognized by the system which is associated with a specific set of grammatical rules.

A functional approach is based on the notion that words or phrases have their underlying functional roles inherent in sentence expressions. In functional analysis, the elements composing a larger structure are identified by the roles they play, while in immediate constituent analysis those are identified by the kinds of elements they are. Functional analyses generally place less emphasis on the order of constituents [91].

Being influenced by Katz and Fodor's semantic theory [31], some researchers emphasize direct use of semantic aspects, or closely integrate the use of both syntactic and semantic information to analyze sentences. This includes Fillmore's case grammar [26], Halliday's systemic grammar [47], Bresnan and Kaplan's lexical-functional grammar [12, 42], and Kay's functional unification grammar [43]. Syntactic analysis using these grammars can be called functional analysis, because the concept of function plays a central role in all these grammars. For example, in systemic grammar a sentence simultaneously embeds the following systems: mood, transitivity, theme, and information [47, 91]. Each system is in turn composed of several functional roles. For instance, the roles GOAL, ACTION, and ACTOR are conveyed in the transitivity system.

Functional analysis has been attracting more researchers recently because it can offer some advantages over the immediate constituent analysis. Many linguists believe that syntactic regularities are stated more naturally and economically within a functional framework. Functional description also makes cross-language generalization more feasible because different languages may well serve the same functions in the process of transmitting ideas. Furthermore, because functional analysis is quite relevant to meaning analysis, it has generally been more readily adopted by the research that emphasizes the integration of syntax and meaning than immediate constituent analysis [91].

As mentioned previously, language use becomes more and more flexible and succinct after successive evolution. However, it is at the cost of proliferating ambiguities in interpretation. The more flexible or more succinct a language is, the more often ambiguity problems may occur. By contrast, artificial languages such as computer languages generally require more strict specifications in their use, and more rigid format and thus more length in their expression.

As long as we are able to resolve the ambiguity problems, use of ambiguous words should not be discouraged. In the first place, we may think that we have to limit the use of the words or structures that may cause ambiguity problems. For communication in daily life, however, nobody would like to use a natural language in the same way as one has to write or

read a computer language, such as FORTRAN, which does not have so many ambiguity problems. As a matter of fact, although all kinds of natural languages in the world have more or less ambiguity problems, people seldom have trouble in using their languages. In order to gain flexibility and succinctness in the use of language, having ambiguity problems can be a justifiable cost [65]. People seem very good at solving such problems, often efficiently and effectively enough. This is because people are capable of using linguistic knowledge coupled with the information about a discourse or about a domain to derive correct interpretations of sentences. Errors, however, do occur.

Use of language can be considered as a cognitive process in which ideas are first represented as symbols or sounds, and then the ideas bound to the symbols or sounds are recovered [43, 91]. English words and punctuation marks in sentences are such representation symbols. In other words, language use can be seen as a process of information transmitting that requires a consistent algorithm for encoding and decoding. Language generation is a process of encoding a set of ideas into a language representation medium, while language understanding is a process of decoding the language representation medium into the other set of ideas. The two sets should be identical semantically, if they are processed according to a consistent algorithm for encoding and decoding. In fact, people who are using the same language must have in mind the same grammars and similar approaches to either generate a sentence or understand a sentence. Now the problems are how we can make use of available information to resolve ambiguities, and how we can recover the original meanings conveyed by the language producers (i.e., speakers or writers).

Each word in a sentence plays some role in the process of making up the ultimate interpretation of the sentence. There are three types of roles a word can play. In general, English words can be divided into two groups: content words and function words [1]. Content words, such as nouns, verbs, adjectives, and adverbs, have meanings or contents associated with them. Words in this group are also called open classes, since the number of such words increases all the time. A word is the simplest form of a sentence constituent. Constituents can be grouped into a larger constituent with a meaning relevant to each of its components. For example, the noun phrase "red books" has a meaning that is intersected by the meanings "objects that are red" and "objects used for recording knowledge in symbols." A nominal phrase is a highly succinct expression that can convey a rich set of meanings associated with each word in the phrase. To indicate "an institute that is doing research on the safety problems that are associated with construction work," for example, the nominal phrase "construction safety research institute" is a succinct substitute.

Function words can perform syntactic functions (or mechanisms) in a sentence, but with no or little semantic contribution to the sentence. Such words include conjunctions (and, or), articles (the, a), demonstratives (this, that), and prepositions (with, in, at). Words in this group are also called closed classes, since the number of such words does not increase. Such words specify structural relationships among the words in a sentence or across adjunct sentences. For example, the word "that" in the sentence "The book that I bought yesterday is gone" is used to structurally connect its following embedded sentence "I bought yesterday" with its preceding word "book." The word "and" can link two predicates in a simple form. For example, the sentence "He wants to see John and Mary" is a succinct expression for the two sentences "He wants to see John" and "He wants to see Mary." Although such words do not contribute meaning to the interpretation of a sentence, they instead offer syntactic functions which allows us to express ideas in a highly succinct format. As such, function words make it possible for English to have features of succinctness and flexibility.

Nonetheless, some English words have characteristics of both content words and function words. This is evidenced by the primary term of a split term, which has two parts: primary term and variable term. Each of these two terms can be a single word or multiple words, as illustrated in Section 2.2.1 by the split term FISCAL YEAR 88. A primary term, like content words, contributes the major or control meaning of its associated split term. Moreover, a primary term is expected to syntactically bind a variable term in order to have a complete set of meaning.

In a sentence a variable term is not necessarily located next to its associated primary term. A variable term can be located right after or right before its associated primary term. A variable term can also be away from the location of its associated primary term. For instance, in the two sentences "He turned the light on" and "He turned on the light," the variable term "on" can be away from and next to its associated primary term "turned." As a matter of fact, English has a lot of split terms. A multiple-word verb, which has a major verb and associated words to perform as a single verb, can be considered as a split term. The verbs such as turn, make, pick, and get serve as primary terms which are expected to bind with other particles such as on, up, off, and out. Note that each of the particles above can be located right after its associated verb or be located after a direct object.

The concept of split terms can also be applied to sentences typically discussed in the case grammar. In the case grammar, a verb can serve as a case header which decides, for example, cases such as direct object, indirect object, location, and instrument. The case

header can also be considered as a primary term which is expected to syntactically bind with other variable terms. For example, in the sentence "He wrote a letter to John with a pencil," the primary term "wrote" is expected to syntactically bind with the variable terms "some object," "to somebody" and "with some instrument."

]

Appendix B

Specification of Terms Used in Expert-MCA

The terms used in Expert-MCA are classified into the following six types: Fieldname, Fieldvalue, Synonym, Rule, Procedure, and Word/Phrase. The specification of them are described as follows.

B.1 Specification of Fieldname

A frame-based format is applied to represent a term. The values in slots of term frames can be a single word, a phrase, a simple sentence, a series of production rules, or a series of procedural statements. A term frame has the term being represented as its name. A term type is also specified in a term frame. Term frames have different slot names for different types of terms. Each term frame has two parts: leading part and property part. The leading part contains the term being represented and its associated term type, while the property part has several elements, each containing a slot name and its associated values. To save its size, the dictionary does not record the slots with empty value.

The terms with the type Fieldname have the following slots: Capces-Name, Capces-Alias, English-Name, Context, Print-Name, Data-Length, File-Segment, Data-Type, Agency-Responsible, Footer, Description, and Comment.

Most of the information stored in Fieldname term frame are copied from CAPCES database. Expert-MCA does not allow the user to delete any Fieldname from the dictionary. The user are not expected to modify the values of all the slots except the four: Print-Name, Context, Footer, and Comment.

The slot Capces-Name stores the field name coded by the developers of CAPCES database. Each data item in CAPCES has a unique field name by which the corresponding data item can be manipulated by using the database query language FOCUS. A field name cannot have more than 12 alphanumerical characters. The slot Capces-Alias is for an alias of the field name. This slot value is optional. Both field name and its alias can be recognized by the CAPCES database management system. A field name also has a English name which is stored in the slot English-Name. A field name is usually an abbreviation or an acronym of its English name. Although English name is not recognized by the database management system, Expert-MCA accepts all of these three representations for a field.

The slot Context contains the information of context or contexts about the corresponding field. This information is important while dealing with complicated queries which often require contexts or keywords to relate to other terms. The slot can have several contexts with the most specific one in the beginning. The information is used in Procedures or Rules. Via the inference on such contextual information in Rules and Procedures, Expert-MCA is able to process knowledge in depth in the semantic analyzer and reasoner. Terms of other term types also have this Context slot.

The slot Print-Name contains a name that will be printed as a column name on reports. The value in this slot is optional and can be modified by the user. If a field needs explanations about the meaning of its coded values, the message that is entered in the slot Footer will be automatically printed as footer on reports.

The value in the slot Data-Length is a number indicating how many characters the data value is. Data-Type has a value either "A," for an alphabetical value, or "N," for a numerical value. The slot File-Segment indicates a name of a segment in which the field resides. The agency who is responsible for the maintenance of the field is stored in the slot Agency-Responsible. The slot Description details the meaning or the use of the corresponding field. The user can also make comments into the slot Comment.

B.2 Specification of Fieldvalue

The terms with the type Fieldvalue only have three slots In-Which-Field, Context, and Comment. The name of a field of which the term is a field value is stored in the slot In-Which-Field. The value for the slot Context is optional, since it will be inherited from its associated field. The user, however, can enter a value if it is possible to have a more specific keyword for this term than that of its associated field. The slot Comment is used to for comments.

In CAPCES database, MA is a field value of the field PRCD (for project code). The user can define this term MA in Expert-MCA (in Teach/Edit Session, see Section 6-5) by entering PRCD in the slot In-Which-Field. Expert-MCA records this information and creates a term frame for MA by adding other appropriate information if necessary. The term frame then is stored into the dictionary with this content:

((MA VALUE))

((In-Which-Field PRCD))

Expert-MCA will make an instance of the associated field as it looks up the dictionary.

B.3 Specification of Synonym

Synonyms only have two slots Synonym-Of and Comment. The slot Synonym-Of stores its synonymous term. The term with a type of Synonym will inherit properties from the term that is indicated in the slot Synonym-Of.

For example, the term frame SMALLER THAN is stored in the dictionary with the content:

((SMALLER THAN Synonym)
((Synonym-Of LT)))

where LT (for less than) is a Word/Phrase.

After looking up the dictionary for the term SMALLER THAN, Expert-MCA will produce the following:

((SMALLER THAN Word/Phrase)
(Part-Of-Speech CONJ)
(Definition LT)
(English-Name LESS THAN)
(Context LOGICAL-OPERATOR)
(Synonym-Of LT)))

The term stored in the slot Synonym-Of can be another Synonym term as well. In this case, the properties of the current term will be inherited from a term which is not a synonym. In doing so, the system is able to dramatically reduce its dictionary size.

B.4 Specification of Word/Phrase

A Word/Phrase term frame has nine slots as follows: Definition, Part-Of-Speech, Context, Variable-Part, Variable-Location, Variable-Type, Variable-Range, English-Name, and Comment. Values in the first two slots are required, while values in the remaining slots are optional.

The slot English-Name is used to store an English Name for the term represented in a

term frame if necessary. The slot **Definition** stores the meaning of the term being represented. The slot is also called "**meaning**" slot. The value in this slot can be a contextual phrase (please refer to Section 5.4.1), an arithmetical manipulation phrase, the term itself, or nothing. The slot value will be converted into an internal representation. It is also a component that helps the semantic analyzer to determine an appropriate meaning for the terms having multiple meanings, or to group the terms in a sentence into new phrases that have meanings equivalent to contextual phrases.

As mentioned in Section 2.2.3, English words usually can be separated into two groups: **content words** and **function words**. The content words such as nouns, verbs, adjectives, and adverbs have meanings or contents associated with them. The function words perform some functions syntactically in a sentence, but with no semantic contribution to the sentence. Nonetheless, some English words, such as the primary terms in split terms (see Section 2.2.1), tend to perform as both content words and function words.

To extract a split term from an input sentence, Expert-MCA requires the user to define its primary term with specifications about its variable terms. Without defining all the military station names starting with FORT, the user can define a term FT by entering STATION IS "Fort ?x" into this slot, where "?" is a symbol indicating a variable to be bound to a term. By having the slot value like this in conjunction with other associated slot values in the same term frame for the term FT, Expert-MCA is able to semantically form a contextual phrase by binding Ft with the term following it in an input sentence. For example, although the word DIXSON is unrecognized by the system, the words "Ft Dixson" in a sentence like "What are the design status for the projects at Ft Dixson for fy 89" will be interpreted as STATION IS "Fort Dixson" in an internal representation for the sentence. A more subtle use of this feature is to extend to multiple variable bindings, such as in defining the term DESIGN PERCENT as a Word/Phrase with the value in its meaning slot: DESIGN PERCENT ?X "D?Y", where ?X is used to bind a logical operator and ?Y is used to bind a number in an input sentence. The variable words in a split term are not necessarily located next to the primary term. They can also be located before the primary term.

The slot **Part-Of-Speech** is used to store a part of speech or a phrase name for the term. When the user first defines a term, he or she may select a value by toggling special keys on the keyboard from a predefined value list: NOUN, NP (for noun phrase), VERB, PREP (for preposition), PP (for prepositional phrase), PRON (for pronoun), DET (for determiner), ADJ (for adjective), ADV (for adverb), and INTE (for interjection). Such information is used in

syntactic analysis. The use for the slot Context is the same as the one in a Fieldname (please refer to Section 5.4.2).

The slot Variable-Part is used to sequentially indicate variable symbols entered in the meaning slot. For example, the slot of the Word/Phrase term DESIGN PERCENT mentioned above has variables ?X and ?Y. The information about the location of the variable words in a sentence relative to the location of the primary term (i.e., the term expecting to bind with other words) is entered in the slot Variable-Location. The slot value can also be selected from a candidate list having values such as: next, previous, next next-2, and previous previous-2. The value NEXT NEXT-2 is for the case of binding with two variable words: first one is located in the input sentence next to the primary term and the second one is located at next two words.

The information about data type and data range of variable words is stored in the slots Variable-Type and Variable-Range, respectively. The slot Variable-Type has values such as: number, string, word, logical-operator, Fieldname, fieldvalue, phrase, undefined, number-or-string, logical-operator number, logical-operator string, and logical-operator word. If multiple sets of values are suitable for a variable word, the user should select the most specific one. In the slot Variable-Range, empty value or N/A means no constraint. To specify a numerical range, the user can enter a pair of lower and upper bound data in a parenthesis. If there is a single variable word to be bound, the numerical values of a range can be entered without a parenthesis.

Here shows a list consisting of major content of a Word/Phrase, DESIGN PERCENT:

```
((DESIGN PERCENT Word/Phrase)
((Definition DESIGN PERCENT ?X "D?y")
(Part-Of-Speech NP)
(Variable-Part ?X ?Y)
(Variable-Location NEXT NEXT-2)
(Variable-Type LOGICAL-OPERATOR NUMBER)
(Variable-Range N/A (0 99))
))
```

By making use of the information stored in the above slots for DESIGN PERCENT, Expert-MCA is able to extract⁵⁵ the phrase "design percent greater than 90" from the sentence "What are the projects with design percent greater than 90," or "design percent less than 10" from "What are the projects with design percent less than 10."

⁵⁵This is done in the lexical analysis module, as described in Section 6.2.3

B.5 Specification of Rule

The construct of Rules is fairly straightforward. The property part of a Rule term can contain Rules as many as needed. Each Rule starts with a flag Rule, followed by its name, and then followed by IF and THEN parts. IF part starts with a flag IF followed by a series of condition statements, while THEN part starts with a flag THEN followed by a series of action statements. To illustrate, the content for Set-Rpt-Rule (rules for setting report content) can be defined as follows:

```
((SET-RPT-Rule Rule)
 (Rule MACOM-1
  (IF (USER IS-A MACOM))
  (THEN (REPORT CONTAINS FY PROG_AMT))))
 (Rule DAEN-1
  (IF (USER IS-A DAEN))
  (THEN (REPORT CONTAINS FY PROG_AMT PROJECT_DESC))))
```

Expert-MCA provides the user with a special editor designed for editing Rules as above. The editor will provide mechanisms for producing parentheses automatically.

B.6 Specification of Procedure

The property part of a Procedure term consists of a slot Part-Of-Speech with its slot value, and a series of procedural statements. Most of procedural statements begin with a command followed by its arguments. There are about ten commands used in Procedure frames. Definition of two Procedures⁵⁶ are shown as follows:

⁵⁶The use of these two are illustrated in Chapter 8.

1. ;;This column is for comments.
SUBMIT PROC ;declare a knowledge procedure **SUBMIT**
PART-OF-SPEECH VERB
DECL FACT X Y Z U ;declare X, Y, Z, U as facts
GET X OBJECT ;get a fact bound to a context **OBJECT**
GET Y AGENCY ;get a fact bound to a context **AGENCY**
GET Z DATE-TYPE ;get a fact bound to a context **DATE-TYPE**
U = DETERMINE USING SUBMIT-RULES; call a rule with input
OUTPUT U

2.
DESIGN-FUNDS PROC ;declare a knowledge procedure
PART-OF-SPEECH NOUN ; **DESIGN-FUNDS**
DECL FACT X ;declare X as a fact
DECL VAR X1 X2 ;declare X1 X2 as numerical variables
DECL FIELD F1 F2 F3 ;declare F1 F2 F3 as temporary fields
IF SCREENFIELDS NOT CONTAIN FY THEN ASK FY
GET X FY ;get a fact bound to a field **FISCAL YEAR**
X1 = X + 1
X2 = X + 2
F1 = CALL DESIGN-FUND1 (X1) ;call procedure **DESIGN-FUND1**
F2 = CALL DESIGN-FUND2 (X2) ;call procedure **DESIGN-FUND2**
F3 = F1 + F2
REMOVE FY FROM SCREENFIELDS ;remove FY from screening fields
APPEND DISTRICT TO SORTFIELDS ;append **DISTRICT** as a sorting field
OUTPUT SUM F3 BY SORTFIELDS FOR SCREENFIELDS

The command **DECL** is used to declare data types, as specified by its first argument, for the variables that starts with its second arguments. A data type for the procedural variables in the frame can be a field, a fact, a simple numerical variable, or a report table. The data type **FIELD** here is used for the variables that are bound to derived or temporary fields. A **FACT** variable is used to represent a fact or field instance available from the input sentence or user profile. The statements starting with the command **IF** are for branching conditional actions.

The command **GET** is to find a fact from the information blackboard by searching for a context which is specified as the third argument in the **GET** statement. The output fact will be bound to its second argument. The command **CALL** is used to trigger another Procedure in conjunction with input arguments if any. The commands **DETERMINE** and **DISCOVER** are used to infer Rule terms by using forward chaining algorithm and backward chaining algorithm, respectively. The commands **REMOVE**, **APPEND** are used to modify printing or sorting items. The commands **OUTPUT** and **REPORT** are used to specify a value or values after calling a Procedure term.

Appendix C

Syntax for FOCUS Query Language

The syntax for FOCUS query language described here is the most important part used for its retrieving data. The reader may be referred to the FOCUS User's Manual [79] for more details.

C.1 Syntax for TABLE Command Statements

A tabular report is requested by the command TABLE, followed by a request statement⁵⁷, and a finish command. A request statement may consist of the following components: format-control, verb-phrase, sorting-phrase, screening-phrase, and control-phrase. In what follows the content between the bracket symbols "[] " is optional. Syntax for a major portion of FOCUS query language is given below. Semantics of these symbols can be referred to Section 6.4 and FOCUS User's Manual. A TABLE command statement looks like this:

```
TABLE FILE <filename>
[<format-control>]
<verb-phrase>
[<sorting-phrase>]
[<screening-phrase>]
[<control-phrase>]
<end-statement>
```

The symbols used above are defined in Backus-Naur Form as follows:

```
<end-statement> ::= END | RUN | QUIT
<verb-phrase> ::= <verb> <and-fieldname>|<verb-phrase> <verb> <and-fieldname>
<verb> ::= LIST | PRINT | COUNT | SUM | ADD | WRITE
<prefix> ::= MAX | MIN | AVE | ASQ | FST | LST | PCT | RPCT |
TOT | SUM | CNT | ALL |
<field> ::= <fieldname> | <prefix>.<fieldname> | COLUMN-TOTAL
<and-fieldname> ::= <field> | <and-fieldname> AND <field> |
<and-fieldname> <field> | OVER <field>
<and-value> ::= <value> | <and-value> AND <value>
```

⁵⁷In some complex cases, it may have multiple request statements.

**<sorting-phrase> ::= BY <fieldname> | BY <by-selector> <fieldname> |
By <fieldname> IN-GROUPS-OF <value> |
By <fieldname> IN-GROUPS-OF <value> TOP <value> |
RANKED BY <by-selector> |
ACROSS <fieldname> |
ACROSS <fieldname> IN-GROUPS-OF <value> |
ACROSS <fieldname> IN-GROUPS-OF <value> TOP <value> |
ACROSS <fieldname> COLUMNS <and-value> |**

**<by-selector> ::= HIGHEST | HIGHEST <integer> | TOP | TOP <integer> |
LOWEST | LOWEST <integer>**

**<screening-phrase> ::= IF <field> <relation> <or-value> |
IF <field> FROM <or-value-to-value> |
IF <field> NOT-FORM <or-value-to-value> |
IF TOTAL <field> <relation> <or-value>**

**<relation> ::= IS | EQ | IS-NOT | NE | IS-FROM | FORM | GE | TO |
LE | IS-MORE-THAN | EXCEEDS | GT | IS-LESS-THAN |
LT | CONTAINS | INCLUDES | EXCLUDES | OMITTS**

<or-value> ::= <value> | <or-value> OR <value>

<value-to-value> ::= <lower-value> TO <upper-value>

<or-value-to-value> ::= <value-to-value> | <or-value-to-value> OR <value-to-value>

<control-phrase> ::= ON <fieldname> <on-selector>

**<on-selector> ::= SUB-TOTAL | SUBTOTAL | PAGE-BREAK | SKIP-LINE |
FOLD-LINE | SUMMARIZE | RECOMPUTE | SUP-PRINT |
NONPRINT | COMPUTE | RECAP | UNDER-LINE | SUBFOOT |
SUBHEAD | REPAGE**

Where

<fieldname> : a field name defined in the data base;
<literal> : a combination of alphabets and numbers;
<value> : a literal or a number;
<integer> : an integer;
<lower-value> : a <value> used as a lower limit;
<upper-value> : a <value> used as a upper limit;
<format-control> : statements to control report format or print
necessary text such as a heading, footing, etc.
[<content>] : the <content> is optional.

C.2 Syntax for DEFINE Command Statements

In FOCUS, temporary data fields can be defined as arithmetical or logical combinations of real fields or other temporary data fields. To define temporary data fields needs to issue the DEFINE command statements as follows:

```
DEFINE FILE filename  
<temp-field-expression>
```

END

The syntax for <temp-field-expression> is expressed in Backus-Naur Form as follows:

```
<temp-field-expression>
 ::= <temp-field>/<field-format> = <expression> ; |
    <temp-field-expression>
    <temp-field>/<field-format> = <expression> ;
<temp-field> ::= <literal>
<field> ::= <fieldname> | <temp-field>
<field-format> ::= A<integer> | I<integer> | D<integer>.<integer>
<arith-operand> ::= + | - | * | / | ** | ( | )
<logic-operand> ::= EQ | NE | GT | GE | LT | LE | AND | OR |
    NOT | CONTAINS | OMMITS
<logic-operation>
 ::= <field> <logic-operand> <value> |
    (<logic-operation>) |
    <logic-operation> OR <field> <logic-operand> <value> |
<expression> ::= <arith-operation> |
    IF <logic-operation> THEN <result> ELSE <result>
<result> ::= <field> | <value> | <arith-operation> | <expression>
```

Where

<fieldname> : a field name defined in the data base;
<literal> : a combination of alphabets and numbers;
<value> : a literal or a number;
<integer> : an integer;
<arith-operation> : a combination of <field>, <value>, and <arith-operand>.

Appendix D

List of Interface Tasks in Expert-MCA

The interface tasks implemented in Expert-MCA are listed, each being with some slot contents, as follows:

1. TOP-TASK

Name: Top-Task
Window-Type: single-selection-menu
Option: ("Overview" "Query" "Report" "Download" "Teach"
"Profile" "Utility" "Logout")
Next-Task: (overview query report download teach
update-profile utility exit-to-Lisp)
Note: This is the Main Menu of Expert-MCA.

2. OVERVIEW

Name: overview
Option: ("Overview about Expert-MCA" "Overview about Query Session"
"Overview about Standard Reports")
Window-Type: single-selection-menu
Next-Task: (Expert-MCA-overview Query-overview standard-report-overview)

3. EXPERT-MCA-OVERVIEW

Name: expert-mca-overview
Window-Type: message-lookup
Message: (read-string-file (get-data-file 'expert-mca-overview))
;; The message is read from a file
;; C:\MCA-TASK\EXPE-MCA.ovw, which is an overview about
;; Expert-MCA. This file is an ASCII flat file
;; and can be edited as wished.
Next-Task: (overview)

4. QUERY-OVERVIEW

Name: query-overview
Window-Type: message-lookup
Message: (read-string-file (get-data-file 'query-overview))

Next-Task: (overview)
Note: Here is an Overview about Query Session in Expert-MCA.

5. STANDARD-REPORT-OVERVIEW

Name: standard-report-overview
Window-Type: message-lookup
Message: (read-string-file (get-data-file 'standard-report-overview))
Next-Task: (overview)
Note: This task is an Overview about Standard Reports Session in Expert-MCA.

6. QUERY

Name: query
Option: ("Input Query" "Modify Prev Queries" "Exec Prev Queries")
Window-Type: single-selection-menu
Next-Task: (Input-Query Modify-Prev-Queries Exec-Prev-Queries)

7. INPUT-QUERY

Name: input-query
Window-Type: text-input
After-Action: (mca1-parse)
Next-Task: (understanding-correction correct-unrecognized-term)

8. MODIFY-PREV-QUERIES

Name: Modify-Prev-Queries
Window-Type: frame-slot-editor
Before-Action: (Collect-previous-queries)
After-Action: (pick-up-the-modified-query ;this is a function name.
mca1-parse) ;this is another function name.
Next-Task: (understanding-correction correct-unrecognized-term)
;;Note: Modifying One of the Previous Queries.

9. EXEC-PREV-QUERIES

Name: Exec-Prev-Queries
Before-Action: collect/exec-prev-query
Option: nil
;;The content of this slot will be filled out by the function
;;Collect/exec-prev-query, which is stored in Before-Action slot.

Window-Type: single-selection-menu
After-Action: (pick-up-selected-exec-prev-query
mca1-parse) ;MCA1-parse will select a next task to run.
Next-Task: (understanding-correction correct-unrecognized-term)
;;Note: Choose a Query to Execute

10. UNDERSTANDING-CORRECTION

Name: understanding-correction
Window-Type: frame-slot-editor
After-Action: (mca1-query-generator)
Next-Task: (Focus-correction)
Note: This task is to correct Expert-MCA's Understanding of the Query if Necessary.

11. FOCUS-CORRECTION

Name: Focus-correction
Window-Type: text-editor
Window-Size: (1 0 78 12) ;nx, ny, width, and height of the outer frame
After-Action: select-target-db ;look for local databases in PC first
Next-Task: (pc-or-mainframe-or-cancel mainframe-or-cancel)
;The order of the above slot content is significant in function
; select-target-db.
Note: This task is to correct FOCUS codes if necessary.

12. MAINFRAME-OR-CANCEL

Name: mainframe-or-cancel
Window-Type: single-selection-menu
Option: ("Mainframe Now" "Mainframe in Batch Later"
"Mainframe in Batch Now" "Cancel"
"Conflict Info with Local Databases")
After-Action: (select-next-task-and-pass-arg-in-mainframe-or-cancel)
Next-Task: (comm-with-mainframe add-mainframe-batch run-mainframe-batch
query conflict-info-w/-pc-db)
;;Note: To Choose One To Run For The Current Query

13. CONFLICT-INFO-W/-PC-DB

Name: conflict-info-w/-pc-db
Before-Action: collect-conflict-info-when-tring-to-access-pc-database
Window-Type: message-lookup
Message: nil

Before-Action: ;the Message: slot content will be updated by the function
slot.
Next-Task: (mainframe-or-cancel)
;;Note: to take a look.

14. COMM-WITH-MAINFRAME

Name: comm-with-mainframe
Window-Type: self-execution
;It executes the function COMM-WITH-MAINFRAME, which
;controls its window specifications.
After-Action: next-task-after-comm-with-mainframe
Next-Task: nil ;assigned by the above function
;;Note: In Communication with Mainframe

15. PC-OR-MAINFRAME-OR-CANCEL

Name: pc-or-mainframe-or-cancel
Window-Type: single-selection-menu
Option: ("PC Databases" "Mainframe Now" "Mainframe in Batch Later"
"Mainframe in Batch Now" "Cancel")
After-Action: (select-next-task-and-pass-arg-in-pc-or-mainframe-or-cancel)
Next-Task: (run-pc-database-now comm-with-mainframe add-mainframe-batch
run-mainframe-batch query)
;;Note: To Choose One To Run For The Current Query

16. RUN-PC-DATABASE-NOW

Name: run-pc-database-now
Before-Action: in-comm-with-pc
Window-Type: message-lookup
Message: (" "
"Summary of the data retrieval.")
Next-Task: (query)
;;Note: just finished a session for data retrieval from local databases.

17. CORRECT-UNRECOGNIZED-TERM

Name: correct-unrecognized-term
Option: ("DEFINE NEW WORD" "EDIT WORD" "IGNORE WORD")
Window-Type: single-selection-menu
After-Action: (pass-unrecognized-term-to-next-task)
Next-task: (new-word/phrase-check edit-word ignore-word)

18. NEW-WORD/PHRASE-CHECK

Name: new-word/phrase-check
Option: ("No" "Yes")
Window-Type: single-selection-menu
After-Action: (update-unrecognized-terms-in-new-word/phrase-check)
Next-Task: (select-term-type-for-unrec enter-phrase)
Note: Is this term part of a phrase?

19. ENTER-PHRASE

Name: enter-phrase
Window-Type: line-text-input
After-Action: (update-unrecognized-terms-in-enter-phrase)
Next-Task: select-term-type-for-unrec
Note: enter the phrase that contains the unrecognized term.

20. EDIT-WORD

Name: edit-word
Window-Type: line-text-input
After-Action: (update-unrecognized-term-in-edit-word)
Next-Task: (correct-unrecognized-term)
Note: enter a correct one.

21. IGNORE-WORD

Name: ignore-word
Window-Type: help-message-lookup
Message: ("You want to ignore the term. Please hit F5 key to proceed.")
After-Action: (update-unrecognized-term-in-ignore-word)
Next-Task: (correct-unrecognized-term)

22. SELECT-TERM-TYPE-FOR-UNREC

Name: select-term-type-for-unrec
Option: ("Field Instance" "Synonym" "Word" "Phrase"); "Procedure Rule")
Window-Type: single-selection-menu
After-Action: (remember-which-type-for-unrec) ;to do with order of OPTIONS.
Next-Task: (define-new-term)
Note: Terms can be defined as one of the following types.

23. REPORT

Name: report
Option: ("Review/Update Reports" "Add Reports" "Delete Reports"
"Execute Reports")
Window-Type: single-selection-menu
Next-Task: (select-Report Add-Report Select-Report select-report)

24. SELECT-REPORT

Name: select-report
Before-Action: (prepare-report-names)
Option: nil ;This will be updated by the function of Prepare-report-list.
Window-Type: single-selection-menu
After-Action: decide-one-report-task
Next-Task: (update-Report Delete-Report Execute-Report)
;; One of the 3 tasks will be selected to execute depending on the input
;; to the current task Select-report.

25. UPDATE-REPORT

Name: update-report
Before-Action: (prepare-report-content)
Window-Type: frame-slot-editor
After-Action: (update-report-to-dictionary)
Next-Task: (report)

26. ADD-REPORT

Name: add-report
Before-Action: (prepare-report-names)
Window-Type: line-text-input
Next-Task: (update-report)

27. DOWNLOAD

Name: download
Option: ("Review/Update Downloading Tasks" "Add Downloading Tasks"
"Delete Downloading Tasks" "Execute Downloading Tasks")
Window-Type: single-selection-menu
Next-Task: (select-download Add-download Select-download select-download)

28. SELECT-DOWNLOAD

Name: select-download
Before-Action: (prepare-download-names)
Option: nil ;This will be updated by the function of Prepare-download-list.
Window-Type: single-selection-menu
After-Action: decide-one-download-task
Next-Task: (update-download Delete-download Execute-download)
;;One of the 3 tasks will be selected to execute depending on the input
;; to the current task Select-download.

29. UPDATE-DOWNLOAD

Name: update-download
Before-Action: (prepare-download-content)
Window-Type: frame-slot-editor
After-Action: (update-download-to-dictionary)
Next-Task: (download)

30. ADD-DOWNLOAD

Name: add-download
Before-Action: (prepare-download-names)
Window-Type: line-text-input
Next-Task: (update-download)

31. TEACH

Name: teach
Option: ("Review/Edit Existing Terms" "Define New Terms")
Window-Type: single-selection-menu
Next-Task: (select-context-for-target-terms select-term-type)

32. SELECT-TERM-TYPE

Name: select-term-type
Option: ("Field Instance" "Synonym" "Word" "Phrase") ;"Procedure Rule")
Window-Type: single-selection-menu
After-Action: (remember-which-type) ;This is to do with order of OPTIONS.
Next-Task: (input-new-term)

33. INPUT-NEW-TERM

Name: input-new-term
Before-Action: store-input-for-input-new-term
title-text: nil ;updated by the above function
Window-Type: line-text-input
After-Action: check-and-combine-text-with-term-type
Next-Task: (define-new-term select-from-existing-term)

34. DEFINE-NEW-TERM

Name: define-new-term
Before-Action: set-define-new-term-title
Window-Type: term-frame-editor
After-Action: update-dictionary-in-define-new-term
Next-Task: (teach select-context-for-target-terms)
;; In the process of defining a term using a HELP-DEFINE
;; hori-menu, we may need to search for a target term, either
;; a fieldname of word/phrase. The function stored in
;; AFTER-ACTION: slot has to identify this situation
;; and thus decides which task has to be executed next.

35. SELECT-FROM-EXISTING-TERM

Name: select-from-existing-term
Option: nil ;will be updated by the task which calls this one.
Window-Type: single-selection-menu
After-Action: reset-select-from-existing-term ;This resets next-task.
Next-Task: (review/edit-existing-term define-new-term)

36. REVIEW/EDIT-EXISTING-TERM

Name: review/edit-existing-term
Before-Action: set-review-term-title
Window-Type: term-frame-editor
After-Action: update-dictionary-in-review/edit-existing-term
Next-Task: (teach)

37. SELECT-CONTEXT-FOR-TARGET-TERMS

Name: select-context-for-target-terms
Before-Action: prepare-context-tree
Window-Type: layered-horizontal-menus
After-Action: remeber-context-for-terms

Next-Task: (select-term-after-context select-term-type)

38. SELECT-TERM-AFTER-CONTEXT

Name: select-term-after-context
Before-Action: prepare-for-select-term-after-context
Window-Type: single-selection-menu
Option: nil ;will be updated by the task which calls this one.
After-Action: prepare-diction-list
Next-Task: (review/edit-existing-term define-new-term)
needed-global-variables:
(*qualified-terms* ; set in function collect-qualified-terms.
find-a-term-through-context-tree-p ;This is defined in FR-EDITOR.
returning-term) ;This is needed in HELP-DEFINE hori-menu.

39. UPDATE-PROFILE

Name: update-profile
Window-Type: single-selection-menu
Option: ("Update/Review Domain Profile" "Update/Review User Profile")
After-Action: (assign-next-task-for-profile)
Note: The above function may reassign a new next task
SELECT-PROFILE-TASK-BY-SYSTEM if the user is System.
Next-Task: (update-domain-profile update-user-profile)

40. UPDATE-DOMAIN-PROFILE

Name: update-domain-profile
Before-Action: prepare-to-update-domain-profile
Window-Type: frame-slot-editor
After-Action: update-for-domain-profile
Next-Task: (top-task)
Note: Review/Edit Domain Profile.

41. UPDATE-USER-PROFILE

Name: update-user-profile
Before-Action: prepare-to-update-user-profile
Window-Type: frame-slot-editor
After-Action: update-for-user-profile
Next-Task: (top-task)
Note: Review/Edit A User Profile.

42. SELECT-PROFILE-TASK-BY-SYSTEM

Name: select-profile-task-by-system
Option: ("Review/Update User Profile" "Add User" "Delete User")
Window-Type: single-selection-menu
Next-Task: (select-user add-user delete-user)

43. SELECT-USER

Name: select-user
Before-Action: (send-existing-users-to-option)
Window-Type: single-selection-menu
Option: nil ;This is updated by the function Prepare-existing-users.
After-Action: (prepare-user-profile-for-a-user)
Next-Task: (update-user-profile)

44. ADD-USER

Name: add-user
Before-Action: (send-and-prepare-existing-users)
Window-Type: line-text-input
After-Action: (prepare-user-profile-for-a-user)
Next-Task: (update-user-profile)

45. DELETE-USER

Name: delete-user
Before-Action: (send-existing-users-to-option)
Option: () ;updated by the above function.
Window-Type: single-selection-menu
After-Action: (update-user-profile-after-deletion)
Next-Task: (select-profile-task-by-system)

Appendix E

List of File Names and File Paths in Expert-MCA

The following is a list of file names and a brief description of content for each file in Expert-MCA.

Directory: Filename: File content:

1. c:\mca-task

config.lsp variables bound with names of subdirectory paths.
tsk-vari.lsp value binding of global variables
window.lsp definitions of window arrangements and attributes
wind-fun.lsp manipulation functions on windows

2. c:\mca-task\basic

tsk-func.lsp Macro functions for reading *.Tsk files
tsk-main.lsp interface task-dispatcher and screen generators
tsk-gene.lsp window generators for various types of tasks
basic.tsk definitions of symbols used in interface tasks
toptask.tsk definitions of interface tasks in the top (or main) menu
toptask.lsp functions used in toptask.tsk, functions for login

3. c:\mca-task\overview

overview.tsk interface tasks in the overview session

4. c:\mca-task\query

query.tsk interface tasks in the query session
query.lsp functions used in query.tsk
lookup.lsp functions for dictionary lookup
parse.lsp functions for parsing queries
reasonco.lsp functions for reasoning by using contextual info.
understa.lsp functions for interpreting queries
focusgen.lsp functions for generating FOCUS codes

5. c:\mca-task\report

report.tsk interface tasks in the standard report session
report.lsp functions used in report.tsk

6. c:\mca-task\download

download.tsk interface tasks in the database downloading session
download.lsp functions used in download.tsk and query.tsk

7. c:\mca-task\teach

teach.tsk interface tasks in the teaching/reviewing session
teach.lsp functions used in teach.tsk

8. c:\mca-task\profile
 - profile.tsk interface tasks in the user profile session
 - profile.lsp functions used in profile.tsk

9. c:\mca-task\utility
 - utility.tsk interface tasks as mca utilities (not implemented)
 - utility.lsp functions used in profile.tsk (not implemented)

10. c:\mca-task\data data files for:
 - esnuser.lsp user profile
 - esnprt.lsp standard reports
 - esndnld.lsp downloading
 - esnchap1.lsp dictionary for CAPCES fields
 - esnchap2.lsp dictionary for CAPCES fields
 - esndic1.lsp dictionary for other terms
 - esndic2.lsp dictionary for other terms
 - *.ovw overview text file - description of different sessions

11. c:\mca-task\commun
 - mca-comm.lsp telecommunications functions used to link to mainframe

12. c:\mca-task\library
 - esnmeng.lsp functions used in a rule's inference engine
 - esnrn0-1.lsp functions for file input and output
 - esnrn0-2.lsp functions for list and string manipulations
 - esnrn0-4.lsp functions for reading data dictionaries
 - esnrn3-2.lsp functions for processing procedural terms
(need to polish this file in the near future)

 - struct.lsp definitions of structures (such as editor, editorfr..)
 - editor.lsp definition of a text editor
 - editorfr.lsp definition of a frame-slot editor
 - se-menu1.lsp definition of a single-selection-menu (vertical)
 - msg.e.lsp definition of a message-echo/lookup template
 - hor-menu.lsp definition of a horizontal-single-selection-menu

13. c:\mca-task\working
 - working.lsp functions used in developing /modifying Expert-MCA
 - esnrnmgmt.lsp functions used to alphabetically sort all the
functions defined in the 45 files of Expert-MCA.
 - mgsorted.fun a name list of all the functions defined in Expert-MCA
it is arranged in an alphabetical order.

References

- [1] Akmajian, A., Demers, R.A., and Hamish, R.M.
Linguistics: An Introduction to Language and Communication.
The MIT Press, Cambridge, MA, 1984.
- [2] Allen, James.
Natural Language Understanding.
The Benjamin/Cummings Publishing Co., Menlo Park, CA, 1987.
- [3] Ashley, David B. and Levitt, Raymond E.
Expert Systems in Construction: Work in Progress.
Journal of Computing in Civil Engineering 1(4), October, 1987.
- [4] Ashley, David B. and Perng Y. H.
An Intelligent Construction Risk Identification System.
In *Sixth International Symp. on Offshore Mechanical and Arctic Engineering.*
Houston, TX, March, 1987.
- [5] Ballard, B., and Tinkham, M.
A Grammatical Framework for Transportable Natural-Language Processing.
Computational Linguistics 10(2), 1984.
- [6] Barr, Avron, and Feigenbaum, Edward A. (editors).
The Handbook of Artificial Intelligence, Vol. 1.
William Kaufmann, Los Altos, CA, 1981.
- [7] Barr, Avron, and Feigenbaum, Edward A. (editors).
The Handbook of Artificial Intelligence, Vol. 2.
William Kaufmann, Los Altos, CA, 1981.
- [8] Barr, Avron, and Feigenbaum, Edward A. (editors).
The Handbook of Artificial Intelligence, Vol. 3.
William Kaufmann, Los Altos, CA, 1981.
- [9] Bates, M., and Bobrow, R.J.
A Transportable Natural Language Interface.
In *Research And Development in Information Retrieval.* ACM, 1983.
The 6th Annual International ACM SIGIR Conference.
- [10] Bever, M. and Ruland, D.
Aggregation and Generalization Hierarchies in Office Automation.
In *Conference on Office Information Systems.* ACM SIGOIS and IEEE CS TC-OA,
March 22-23, April, 1988.
- [11] Bobrow, Daniel G.
Natural Language Input for a Computer Problem Solving System.
In Marvin Minsky (editor), *Semantic Information Processing*, pages 133-215. The
MIT Press, Cambridge, MA, 1967.

- [12] Bresnan, Joan W.
A Realistic Transformational Grammar.
In Morris Halle, Joan W. Bresnan, and George A. Miller (editor), *Linguistic Theory and Psychological Reality*, pages 1-59. The MIT Press, Cambridge, MA, 1978.
- [13] Brisson, Francoise.
Knowledge Based Expert System for Planning and Monitoring construction Projects.
Master's thesis, Civil Engineering Department, Massachusetts Institute of Technology, June, 1987.
- [14] Brodie, M.L., and Mylopoulos, J. (editor).
On Knowledge Base Management Systems.
Springer-Verlag, New York, 1986.
- [15] Carbonell, J. G., Cullingford, R. D., and Gerschman, A. V.
Knowledge-Based Machine Translation.
Technical Report Research Report #146, Computer Science Department, Yale University, New Haven, CT, 1978.
- [16] Charniak, E., and McDermott, D.
Introduction to Artificial Intelligence.
Addison-Wesley Publishing Co., Reading, MA, 1985.
- [17] Chen, Peter .P.S.
The Entity-Relationship Model - Toward a Unified View of Data.
ACM TODS 1(1), March, 1976.
- [18] Chen Frank H.S.
Expert-MCA: An Expert System for CAPCES Database.
Master's thesis, Civil Engineering Department, Massachusetts Institute of Technology, February, 1988.
- [19] Chemeff, Jonathan.
Automatic Generation of Construction Schedules from Architectural Drawings.
Master's thesis, Civil Engineering Department, Massachusetts Institute of Technology, June, 1988.
- [20] Chung, Se-Hack.
The Expert System for Resource Constrained Scheduling.
Master's thesis, Civil Engineering Department, Massachusetts Institute of Technology, June, 1987.
- [21] Cullingford, R.E.
Natural Language Processing - A Knowledge Engineering Approach.
?, ?, 1986.
- [22] Date, C.J.
An Introduction to Database Systems.
Addison-Welsley, Reading, MA, 1986.

- [23] De La Garza, Jesus M. and Ibbs, C. William.
A Knowledge Engineering Approach to the Analysis and Evaluation of Schedules for Mid-Rise Construction.
Technical Report, Construction Engineering and Management, Civil Engineering Department, the University of Illinois at Urbana-Champaign, Champaign, IL, July, 1988.
Civil Engineering Studies, Construction Research No. 23.
- [24] Dorsey, Robert W.
Managerial and Social Problems of Robotization in the United States of America.
In *Fourth International Symposium on Robotics and Artificial Intelligence in Building Construction.* Building Research Station - Technion I.I.T., Haifa, Israel, June, 22-25, 1987.
- [25] Erman, L., Hayes-Roth, F., Lesser, V., and Reddy, D.
The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty.
ACM Computing Surveys 12(2):213-253, 1980.
- [26] Fillmore, Charles.
The Case for Case.
In Emmon Bach, and Robert T. Harms (editor), *Universals in Linguistic Theory*, pages 1-90. Holt, Rinehart, and Winston, Chicago, 1968.
- [27] Finn, G. A. and Reinschmidt, K. F.
Expert Systems in an Engineering-Construction Firm.
In Kostem, C.N. and Maher, M.L. (editor), *Expert Systems in Civil Engineering.* American Society of Civil Engineers, New York, NY, 1986.
- [28] Garvey, A., Hewett, M., Johnson, M., Schulman, R., and Hayes-Roth B.
BB1 User Manual: How to Implement a System in BB1.
Technical Report, Knowledge Systems laboratory, Stanford University, Stanford, CA. October, 1986.
- [29] Ginsparg, J.M.
A Robust Portable Natural Language Data Base Interface.
In *Conference on Applied Natural Language Processing*, pages 25-30. Association for Computational Linguistics, 1983.
- [30] Grosz, B, Appelt, D.E., Martin, P. and Pereira, F.
TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces.
Technical Report 358, Artificial Intelligence Center, SRI International, Menlo Park, California, Aug., 1985.
- [31] Harris, Mary Dee.
Introduction To Natural language Processing.
Reston Publishing Co., Reston, VA, 1985.
- [32] Hendrix, G.G., Sacerdoti, E., Sagalowicz, D., and Slocum, J.
Developing a Natural Language Interface to Complex Data.
ACM Transactions on Database Systems 3(2):105-147, 1978.

- [33] Hendrixson, Chris, Zozaya-Gorostiza, Carlos, Rehak, Daniel, Baracco-Miller, Eduardo, and Lim, Peter.
Expert System for Construction Planning.
Journal of Computing in Civil Engineering 1(4), October, 1987.
- [34] Hindy, Ayman M.
Expert-MCA: A Knowledge-Based Natural Language Interface to CAPCES database.
Master's thesis, Civil Engineering Department, Massachusetts Institute of Technology, February, 1987.
- [35] Howard, H.C. and Rehak, D.R.
Interfacing Databases and Knowledge Based Systems for Structural Engineering Applications.
Technical Report EDRC-12-06-86, Design Research Center, Carnegie-Mellon University, November, 1986.
- [36] Howard, H.C. and Rehak, D.R.
KADBASE - A Prototype System-Database Interface for Integrated CAE Environments.
In *Proceedings of American Association for Artificial Intelligence*, pages 804-808. AAAI, 1987.
- [37] Hull, Richard and King, Roger.
Semantic Database Modeling: Survey, Applications, and Research Issues .
ACM Computing Surveys 19(3), September, 1987.
- [38] Ibbs, C. Williams.
Future Direction for Computerized Construction Research.
Journal of Construction Engineering and Construction 112(3), September, 1986.
- [39] Israel, David.
Notes on Inferences: A Somewhat Skewed Survey.
On Knowledge Base Management Systems.
Spring-Verlag, New York, NY, 1986, pages 98-109.
- [40] Ivan, John N.
CAPEX: A Cafeteria Architectural Planning Expert System.
Master's thesis, Civil Engineering Department, Massachusetts Institute of Technology, February, 1988.
- [41] Jakobson, G., Lafond, C., Nyberg, E. and Piatetsky-Shapiro, G.
An Intelligent Database Assistant.
IEEE Expert :65-79, Summer, 1986.
- [42] Kaplan, Ronald M.
On Process Models for Sentence Analysis.
In Donald A. Norman, David E. Rumelhart, and the LNR Research Group (editor),
Explorations in Cognition, pages 117-135. Freeman, San Francisco, CA, 1975.

- [43] Kay, Martin.
Parsing in Functional Unification Grammar.
In David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky (editor), *Natural Language Parsing*, pages 251-278. Cambridge University Press, Cambridge, England, 1985.
- [44] Kerschberg, L. (editor).
Expert Database Systems.
The Benjamin/Cummings Publishing Co., 1986.
- [45] Kim, Simon S.
Survey of the State-of-the-Art Expert/Knowledge Based Systems in Civil Engineering.
Technical Report Special Report P-87/01, CERL, US Army of Corps of Engineers, October, 1986.
- [46] Kostem, C.N. and Maher, M.L. (editor).
Expert Systems in Civil Engineering.
American Society of Civil Engineers, New York, NY, 1986.
- [47] Kress G. R.
Holiday: System And Function In Language.
Oxford University Press, London, England, 1976.
- [48] Lee, Jintae and Malone, Thomas W.
How Can Groups Communicate When They Use Different Languages? Translating between Partially Shared Type Hierarchies.
In *Conference on Office Information Systems*. ACM SIGOIS and IEEE CS TC-OA, March 22-23, April, 1988.
- [49] Levitt, R. E., and Kunz, J. C.
Using Artificial Intelligence Techniques to Support Project Management.
AI EDAM 1(1), 1987.
- [50] Levitt, R. E., Kunz, J. C., and Kartam, N. A.
Using Artificial Intelligence Techniques for Automated Planning and Scheduling.
In *Fourth International Symposium on Robotics and Artificial Intelligence in Building Construction*. Building Research Station - Technion I.I.T., Haifa, Israel, June, 22-25, 1987.
- [51] Levitt, R. E., Kartem, N. A., and Kunz, J. C.
Artificial Intelligence Techniques for Generating Construction Project Plans.
Journal of Construction Engineering and Management 114(3), September, 1988.
- [52] Li, Qing and Mcleod, Dennis.
Object Flavor Evolution in an Object-Oriented Database System.
In *Conference on Office Information Systems*. ACM SIGOIS and IEEE CS TC-OA, March 22-23, April, 1988.
- [53] Logcher, R.D., Hindy, A., and Wang, M.T.
Expert-MCA, A Knowledge Based Natural Language Interface To CAPCES Database.
Technical Report CCRE-87-1, Civil Engineering Department, MIT, January, 1987.

- [54] Logcher, R.D.
Adding Knowledge Based Systems Technology to Project Control Systems.
In Ibbes, C. William and Ashley, David B. (editor), *Project Controls: Needs and Solutions, Proceeding of a Specialty Conference, Lincolnshire, IL*, pages 88-100. ASCE, New York, 1987.
- [55] Madnick, Stuart E. and Wang, Y. Richard.
Logical Connectivity: Applications, Requirements, and An Architecture.
In Richard Wang and Stuart Madnick (editor), *Connectivity among Information Systems*, pages 37-51. Sloan School of Management, M.I.T., Cambridge, MA, 1988.
Composite Information Systems (CIS) Projects.
- [56] Madnick, Stuart E. and Wang, Y. Richard.
A Framework of Composite Information Systems for Strategic Advantage.
In Richard Wang and Stuart Madnick (editor), *Connectivity among Information Systems*, pages 6-21. Sloan School of Management, M.I.T., Cambridge, MA, 1988.
Composite Information Systems (CIS) Projects.
- [57] Maida, Anthony S. and Shapiro, Stuart C.
Intensional Concepts in Propositional Semantic Networks.
Readings in Knowledge Representation.
Morgan Kaufmann Publisher Co., Los Altos, CA, 1985.
- [58] Melle, William.
The Structure of the MYCIN system.
Rule-Based Expert Systems.
Addison Wesley Publishing Co., Reading, MA, 1984, pages 67-77.
- [59] Mohan S.
Expert Systems Technology in the Domain of Construction.
In *Fourth International Symposium on Robotics and Artificial Intelligence in Building Construction*. Building Research Station - Technion I.I.T., Haifa, Israel, June, 22-25, 1987.
- [60] Nii, H.P.
Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures, Part 1.
The AI Magazine 7(2):38-53, Summer, 1986.
- [61] Nii, H.P.
Blackboard Systems: Blackboard Application Systems, Blackboard Systems form a Knowledge Engineering Perspective.
The AI Magazine 7(3):82-106, August, 1986.
- [62] Paulson, Boyd C.
Automation and Robotics for Construction.
Journal of Construction Engineering and Management 111(3), September, 1985.

- [63] Peckham, Joan and Maryanski, Fred.
Semantic Data Models.
ACM Computing Surveys 20(3), September, 1988.
- [64] Perrault, C.R. and Grosz, B.J.
Natural Language Interfaces.
Technical Report 393, Artificial Intelligence Center, SRI International, Menlo Park,
CA, Aug., 1986.
- [65] Samad, Tariq.
A Natural Language Interface For Computer-Aided Design.
Kluwer Academic Publishers, Boston, MA, 1986.
- [66] Schank, Roger C.
Conceptual Dependency: A Theory of Natural Language Understanding.
Cognitive Psychology 3(4), 1972.
- [67] Schank, Roger C. and Abelson, R.
Scripts, Plans, Goals, Understanding.
Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.
- [68] Schank, Roger C, and Shwartz Steven P.
The Role of Knowledge in Natural Language Systems.
Applications in Artificial Intelligence.
Petrocelli Books, Inc., Princeton, NJ, 1987.
- [69] Scott, A.C., Clancey, W.J., Davis, R., and Shortliffe, E.H.
Methods for Generating Explanations.
In Buchanan, B.G., and Shortliffe, E.H. (editor), *Rule-Based Expert Systems*, pages
338-362. Addison Wesley Publishing Co., Reading, MA, 1984.
- [70] Shieber, S.M.
The Design of a Computer Language for Linguistic Information.
In *International Conference on Computational Linguistics*, pages 362-366.
Association for Computational Linguistics, 1984.
- [71] Slocum, A. H., L. Domsetz, D. Levy, B Schena, and A. Ziegler.
Construction Automation Research at the Massachusetts Institute of Technology.
In *Fourth International Symposium on Robotics and Artificial Intelligence in
Building Construction.* Building Research Station - Technion I.I.T., Haifa, Israel,
June, 22-25, 1987.
- [72] Smith, J.M.
Expert Database Systems: A Database Perspective.
In Kerschberg, L. (editor), *Expert Database Systems.* The Benjamin/Cummings
Publishing Co., 1986.
- [73] Stillings, Neil A., Feinstein, Mark H., Garfield, Jay L., Rissland, Edwina L.,
Rosenbaum, David A., Weisier, Steven E., and Baker-Ward Lynne.
Cognitive Science - An Introduction.
The MIT Press, 1987.

- [74] Templeton, M., and Burger, J.
Problems in Natural-Language Interface to DBMS with Examples from EUFID.
In *Conference on Applied Natural Language Processing*, pages 3-16. Association for Computational Linguistics, 1983.
- [75] Thompson, F.B., and Thompson, B.H.
Practical Natural Language Processing: The REL system prototype.
In Rubinnff, M., and Yovitis, M.C. (editor), *Advances in Computers*, pages 109-168. Academic Press, New York, NY, 1975.
- [76] Tommelein, I. C., Levitt, R. E. and Hayes-Roth B.
Using Expert Systems for the Layout of Temporary Facilities on Construction Sites.
In *CIB W-65 Symp.*. London, UK, September, 1987.
- [77] Tucker, Richard L.
Perfection of the Buggy Whip.
Journal of Construction Engineering and Management 114(2), June, 1988.
- [78] Tueni, Michel, Li, Jianzhong, and Fares, Pascal.
AMS: A Knowledge-Based Approach to Tasks Representation, Organization and Coordination.
In *Conference on Office Information Systems*. ACM SIGOIS and IEEE CS TC-OA, March 22-23, April, 1988.
- [79] Tymshare.
FOCUS User's Manual, Release 4.5.
Technical Report, Information Builders, Inc., New York, 1984.
- [80] US Army Corps of Engineers.
PAX CAPCES Training Manual.
Technical Report, US Army Corps of Engineers, 1984.
- [81] Vassiliou, Y.
Knowledge Based and Database Systems: Enhancements, Coupling or Integration?
In Brodie, M., and Mylopolous, J. (editor), *On Knowledge Base Management Systems*, pages 87-91. Spring-Verlag, New York, NY, 1986.
- [82] Walters, John R. and Nilsen, Norman R.
Crafting Knowledge-Based Systems.
John Wiley and Sons Co., New York, 1988.
- [83] Waltz, D. L.
An English Language Question Answering System for a Large Relational Database.
Communications of the Association for Computing Machinery 21(7):110-122, 1978.
- [84] Wang, Y. Richard, Madnick, Stuart E., and T. K. Wong.
Concept Agents In CIS/TK: A Tool Kit for Composite Information systems.
In Richard Wang and Stuart Madnick (editor), *Connectivity among Information Systems*, pages 80-94. Sloan School of Management, M.I.T., Cambridge, MA, 1988.
Composite Information Systems (CIS) Projects.

- [85] Wang, Y. Richard and Madnick, Stuart E.
Connectivity among Information Systems.
In Richard Wang and Stuart Madnick (editor), *Connectivity among Information Systems*, pages 22-36. Sloan School of Management, M.I.T., Cambridge, MA, 1988.
Composite Information Systems (CIS) Projects.
- [86] Warszawski, Abraham and Sangrey, Dwight A.
Robotics in Building Construction.
Journal of Construction Engineering and Management 111(3), September, 1985.
- [87] Waterman, D. A.
A Guide to Expert Systems.
Addison-Wesley, Reading, MA, 1986.
- [88] Webber, Bonie L. and Nilsson Nils J. (editor).
Readings In Artificial Intelligence.
Tioga Publishing Co., Palo Alto, CA, 1981.
- [89] Weizenbaum, J.
ELIZA - A Computer Program for the Study of Natural Language Communication between Man and Machine.
CACM, Vol. 9 :36-45, 1966.
- [90] Wilensky, R.
Understanding Goal-Based Stories.
Technical Report Research Report #140, Computer Science Department, Yale University, New Haven, CT, 1978.
- [91] Winograd, Terry.
Language As a Cognitive Process, Vol. 1: Syntax.
Addison-Wesley, Reading, MA, 1983.
- [92] Winston, Patrick H.
Artificial Intelligence.
Addison-Wesley, Reading, MA, 1984.
- [93] Winston, P. H. and Horn, B. K. P.
LISP.
Addison-Wesley, Reading, MA, 1984.
- [94] Winston, Patrick H.
Learning Structural Descriptions from Examples.
Readings in Knowledge Representation.
Morgan Kaufmann Publisher Co., Los Altos, CA, 1985.
- [95] Woods, William A.
Progress In Natural Language Understanding: An Application To Lunar Geology.
In *Conference Proceedings 42*, pages 441-450. IFIPS, 1973.

- [96] Woods, William A.
Knowledge Base Retrieval.
On Knowledge Base Management Systems.
Spring-Verlag, New York, NY, 1986, pages 179-195.