CREATING A DATA DICTIONARY FROM A REQUIREMENTS SPECIFICATION

by

BETH HUHN HOFFMAN

B.A., Bradley University, 1978

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

Approved by:

David A. Gustafson

Major Professor

CONTENTS

1. INTRODUCTION AND LITERATURE SURVEY..... 1
1.1 Overview..... 1
1.2 Roles of the Data Dictionary..... 3
1.2.1 Data Administration..... 4
1.2.2 Corporate Data Resource Management..... 4
1.2.3 Software Development..... 5
1.3 Data Dictionary Contents 9
1.4 Types of Data Dictionary Systems..... 12
1.5 Summary..... 13

2. REQUIREMENTS..... 15
2.1 Overview..... 15
2.2 General Requirements..... 15
2.3 Specific Requirements..... 17
2.3.1 Input Requirements..... 17
2.3.2 Output Requirements..... 18
2.3.3 Functional Requirements..... 19

3. DESIGN..... 25
3.1 Design Strategy..... 25
3.2 Data Dictionary Creation..... 26
3.3 Interactive Update Capability..... 28
3.4 Report Generation..... 30

4. IMPLEMENTATION..... 31
4.1 Implementation Details..... 31
4.2 Test Strategy..... 32
4.3 User's Guide..... 32

5. CONCLUSIONS..... 37
5.1 Evaluation of the Implementation..... 37
5.2 Extensions..... 38

6. ACKNOWLEDGEMENTS..... 41

REFERENCES..... 42

APPENDIX A E-R-A Specification Format..... 45

APPENDIX B Data Dictionary Format..... 55

APPENDIX C Sample Report..... 60

APPENDIX D User's Guide..... 64

APPENDIX E Source Code Listings..... 72

LIST OF FIGURES

Figure 1. Structured Analysis Entity Types and Attributes..... 11

Figure 2. Data Element Keyword Template..... 20

Figure 3. Composite Data Item Keyword Template..... 20

Figure 4. Data Flow Analysis of the Data Dictionary Tool..... 21

Figure 5. Attribute Keyword Mappings..... 22

Figure 6. Hierarchy Diagram: Overview of the Data Dictionary Tool..... 26

Figure 7. Hierarchy Diagram: Data Dictionary Creation..... 27

Figure 8. Hierarchy Diagram: Update Capability..... 28

Figure 9. Data Dictionary Commands..... 33

1. INTRODUCTION AND LITERATURE SURVEY

1.1 Overview

This report describes a data dictionary tool that provides an automated facility for capturing data items (along with their attributes) from a requirements specification, for completing the definitions interactively, and for reporting information from the created data dictionary.

The conceptual view of what is important for developing good software has changed over the last two decades from a procedural-oriented view to one that is concerned with understanding data. Early in this time frame, there was a great deal of interest in the logic of programs and in program instructions to execute the intended logic [MA82b]. The emphasis was on improving software development by developing more powerful programming languages. Aside from such language-processing tools as assemblers and compilers, most developers had little in the way of automated support for their efforts. Even non-automated tools of the time, such as flowcharts, concentrated on helping the developer to define the program's logic.

As software developers attempted to construct systems that were increasingly more complex, the problems of poor software quality and extensive development and maintenance costs became more apparent. The search for solutions led to the development of modern software development techniques such as structured programming and top-down design and to the identification of the software life cycle to describe the phases of the software development process [WA81, MA82b]. Although the level of quality and reliability improved, there was still a need

for further improvements.

The problem of developing good software has recently come into focus as a problem of understanding data. Current software engineering methodologies are becoming more data-oriented and less process-oriented [MS80]. Requirements analysis methodologies such as SADT [RO77] and structured analysis [DE78] are data flow approaches, primarily concerned with defining data and functions to operate on that data.

In the final evaluation, the transformation of the data from input to desired output is the real measure of the software's effectiveness. This is important in all types of software. Even real-time systems must perform data transformations accurately, although they do have stringent performance requirements as well.

The need for accurately defined data increases with the size and complexity of the software project. Definitions of the data must be accurately communicated to all involved with the project's development and must be consistent across all modules of the software product. To do so, the data definitions must reside in a central and easily accessible place.

A data dictionary is a repository of information about data [MA76]. Comparable to a dictionary for a language, a data dictionary contains the name of each data item, its definition, and perhaps information about its origin and usage. Pertinent entities to be defined in a data dictionary may include all system components such as data items, data structures, files, databases, processes, reports, and even non-computer based entities. The data dictionary contains only meta-level information about the named entity (for example, Employee-Name); it

does not contain actual instances of the data entity (that is, all existing employee names). [NA80, TE79] The data dictionary serves as a central reference point for descriptions of all the data entities within an organization to ensure consistency and control of the definitions.

The term "data dictionary", or more often "data dictionary system", is generally understood also to include the set of procedures used to build and maintain the contents of the data dictionary. Essentially, these procedures include facilities for entering and modifying data definitions, and a means for reporting the contents. A data dictionary system may include editing software to generate the input required by compilers or other software packages and may include computer applications to perform such activities such as consistency control and change impact analysis [BC77]. The data dictionary, therefore, is a support tool used to record, store, and process information for all concerned with the management, use, and control of data.

1.2 Roles of the Data Dictionary

Software professionals and managers have come to expect data dictionary systems to play a number of roles [LE79]. As a central repository of current information about an organization's data, a data dictionary applies to a wide range of management and technical tasks involving the control and use of data. In fact, one of the "selling points" to an organization considering a data dictionary system is its ability to provide different features to different users [VA82]. The areas in which a data dictionary can play an important role include data administration, corporate data resource management, data standardization, and software development.

1.2.1 Data Administration

The most widely recognized role of the data dictionary system has been as a tool for data administration, especially in a database management system (DBMS) environment. The data administrator's primary responsibilities are to control data definition and its use in all applications, and to ensure that new applications use existing definitions of data, if possible [VA82]. A data dictionary system is an excellent tool for these functions. A data dictionary provides a means for recording and reporting consistent definitions of data for either a DBMS or non-DBMS environment. The data dictionary system can also act as a support tool for the data administrator for monitoring changes by showing the impact of proposed changes on files, databases, programs, and reports.

Although a data dictionary is useful in a non-DBMS environment, it is generally considered essential for good management of a database environment [MA83, LE79]. It is regarded as the database administrator's most important tool: it is used by the database administrator during database planning, design, development, operation, and management [LE79].

1.2.2 Corporate Data Resource Management

With the recognition of data as a valuable *corporate* resource as well as the increasing volume of information in a corporation, the data dictionary system has become an important tool for data resource management and data standardization. This role is an extension of data administration beyond the data processing department to cover the entire corporate data resource.

A data dictionary system controls the corporate data resource by standardizing the definition of data, by controlling access to and usage of the data, and by maintaining accurate information about it [LE79]. Redundant data definitions can be eliminated and the utilization of existing data resources can be improved by analyzing the contents of the data dictionary for redundancy and consistency. Standards can be imposed on both the contents and usage of the data dictionary. It can enforce security safeguards against accidental or deliberate unauthorized use of confidential data [VA82] and thereby protect the corporate resource.

In addition to its primary role of controlling and protecting the corporate resource, the data dictionary system has other uses of interest to management. It can report information about existing corporate data and the procedures which access it across organizational and application boundaries. This improves communications between these units and can reduce the amount of data redundancy in the organization.

1.2.3 Software Development

More recently, the data dictionary has been identified as having a valuable role as a support tool for the entire software development life cycle. In fact, some view the data dictionary system as the *prime* support mechanism for the activities during the various phases of system development [LE79]. Phases of the software development cycle include requirements analysis, design, development, documentation, maintenance, and management.

The role of the data dictionary in the requirements analysis phase of software development is to keep track of the data definitions for the

analyst and provide a central repository of that information. During the requirements phase, information is collected about data entities, their relationships to each other and to processes, and their attributes [CH80]. Descriptions of these items can be stored in the data dictionary and then used by other developers to prevent redundant definitions and to ensure consistency among the individual efforts. The descriptions in the data dictionary can be used in conjunction with analysis and design software in order to analyze various components of the system [NA80] for consistency and completeness. For example, analysis might show that some elements identified for the system are never used. The specification can then be amended to use these elements as originally intended or to eliminate them, depending on the user's requirements. In this manner, the data dictionary helps to validate the requirements document [GU79]. Changes made and recorded in this central repository are thereby able to be easily and accurately communicated to other members of the development team.

The data dictionary can also serve as a support mechanism for other requirements tools, such as data flow diagrams [DE78]. By providing a place to store detailed and precise definitions about the components of the diagrams without cluttering the diagrams themselves, the data dictionary becomes an integral part of the specification. The diagrams together with the data dictionary are used to capture and document the user's requirements and to provide a feedback mechanism to the user. They also serve as a communication mechanism among other analysts and developers.

During the design phase, the data dictionary is a valuable aid for communicating information about data definitions, data processes that

use the data, and their interrelationships [MS80] among many developers. It carries over the logical views of the data entities from the requirements phase to provide a foundation for the design of data structures and functions, and to provide a repository for recording the additional details to be used in the implementation. A data dictionary is an especially important tool in database design. An interactive data dictionary system was developed by [GU81] as a tool specifically to support a methodology for logical database design. This tool provides a software environment to support the designer through the steps of the methodology: first, generation of local user views and then generation of the global database view by interactively integrating the local views and resolving inconsistencies.

Besides passively carrying over logical data definitions from the requirements to the design phase, a data dictionary has been used to *automate* bridging the gap between these two phases of the software development cycle. The Software Workbench System (SWB) utilizing a data dictionary was developed by members of a Japanese corporation to address the problem of bridging the discontinuity between the requirements and design specifications [MA80]. One of the major objectives of the SWB methodology was to provide *computer-aided* support over each life cycle step with explicit consistency between adjoining steps. To meet this objective, a basic component of their system is a comprehensive data dictionary containing information about all system components in terms of objects and relationships. Information about the objects (data and functions) identified during the requirements modeling procedure and the design modeling procedure are stored in the data dictionary in the form of relations. Names, types, text, and attributes for all objects and relationships between each object are

recorded.

The data dictionary's role in the implementation phase is to provide the input for data descriptions in the source code. It provides the programmer with meta-level data about data elements, data structures, and file or database formats. For example, in the case of a database, the data dictionary could include a description of the schema, subschemas, security requirements, and integrity rules [TE79]. The information needed to produce the source code's data descriptions may be extracted manually or via automated techniques, but there is more interest recently in the latter, especially with the advent of fifth generation computer systems. One of the objectives of the fifth generation project is to lessen the burden of software generation [MO81]. Automated facilities to generate data descriptions for applications and module generators to produce source code [CR83] are designed to minimize the time and effort to turn concepts into executable code. The generation of programmer's data is one of the most tangible payoffs of a data dictionary system: it saves time and money by automating the manual process and, at the same time, enforces the correct use of data which can avoid expensive debugging time later [MA83].

Their different roles and uses can influence the scope of the contents of data dictionaries. The simplest system may hold only enough information to document, for example, COBOL file structures [BC77] or the data items used in a requirements specification. A more complex data dictionary could contain database schemas and subschemas, relationships, owners/users, programs/modules, systems, reports, manual tasks, and user-defined entity types [CU81], in essence, all of an

organization's data resources. A data dictionary, therefore, can benefit an individual project, a group of projects, or an entire organization.

1.3 Data Dictionary Contents

According to Aristotle, a definition is a description consisting of *genus* and *differentia* [cited in DE78]. The genus establishes the class that contains the word being defined and differentia refers to the set of characteristics that distinguish that entity from all other members of the class.

This approach to defining terms can be applied to the definitions contained in a data dictionary. To define an entity, then, one must determine the class to which it belongs and then establish a set of characteristics or attributes which make it clear which member of that class is meant.

A data dictionary usually provides for a prescribed set of classes or entity types for the items being defined. These classes are used to characterize the types of entities to be defined in the dictionary. The entity types may vary according to the data dictionary system and to the application. For example, a data dictionary used to support a data flow diagram approach to structured analysis such as DeMarco's [DE78, MS80] would have entity types defined to correspond to the diagram's components. These components are data flows, data stores or files, data elements, and processes[1]. Items of the first two entity

---

1. According to this methodology, a data flow is a pipeline over which data of known composition is transmitted. A file is a time-delayed repository of data. A process is a transformation of incoming data

types may be defined in terms of other data items or data elements. A data element, however, is a "primitive" in the sense that it cannot be defined in terms of other data items. It would be defined only in terms of possible values or range of values. Some data dictionary systems provide the capability for user-defined entity types in addition to the prescribed set [CU81].

Each entity type has a prescribed set of attributes. These attributes make up the set of distinguishing characteristics to describe an entity of the given type. Typical attributes used in a data dictionary to describe a data entity include name, type, structure, description, and possible values or range of values. The set of attributes which are appropriate for defining a data entity may vary with the type of the entity, and also with the intended use of the data dictionary. For the data flow diagram example mentioned above, the attributes to be defined for each class are shown in Figure 1.

The attributes to be used for each entity type are defined by the author of the methodology as the essential characteristics for describing the components of the data flow diagrams at a logical level. Some of the attributes are appropriate to more than one of the entity types. For example, a "name" is required to identify each entity and to associate it with the diagrams. "Composition" contains the actual definition of the data flows and data stores which are usually composed of sub-data items. The DeMarco methodology uses relational operators to express the composition of the data item. If other names are used

---

flow(s) into outgoing data flow(s). [DE78]

1. Data flow

   - name
   - aliases
   - composition (the actual definition)
   - notes (other pertinent information)

2. Data element

   - name
   - aliases
   - values and meanings
   - notes

3. File or data store

   - name
   - aliases
   - composition
   - organization (which type of access method is used; name of data element used as key)
   - notes

4. Process

   - name
   - process number (corresponding to the number on the data flow diagram)
   - description (structured English or pseudo-code)

Figure 1. Structured Analysis Entity Types and Attributes

for the same piece of data, perhaps by different user groups, the "aliases" attribute can record that fact without requiring duplicate definitions.

A data dictionary that is used throughout the project life cycle, especially as the primary input to source code data descriptions, would contain physical attributes, such as size and type, as well as the logical definitions.

1.4 Types of Data Dictionary Systems

There are two principal ways to classify the capabilities and implementations of current data dictionary systems [NA80, MA83]. The first scheme is according to the ability to provide descriptions about the data entities to other software. Along this basis, the data dictionary system would be called either passive or active. The second way to classify a data dictionary system is according to its dependence on other software to perform its functions, that is, either stand-alone or dependent. These two classification schemes are not necessarily orthogonal.

A **passive** data dictionary system is used to enter and retrieve descriptions about data entities. With a passive data dictionary system, definitions of the same data will exist concurrently in other software, such as application programs. Changes in the data dictionary definitions do not automatically cause changes in the corresponding definitions in other software, nor vice versa. This type of system, therefore, does not automatically enforce control of the data.

An **active** data dictionary system provides the only source of data entity descriptions for other components in the software development environment. These other components can include compilers, database management systems, and automatic code generation programs. An active data dictionary system enforces data standardization and usage throughout the organization.

A **stand-alone** data dictionary system is one that is self-contained. In other words, it does not depend on other software such as a database management system to perform its functions. It has its own maintenance

and reporting functions. A stand-alone data dictionary system may be either passive or active.

A data dictionary system that is specifically designed to work together with another general purpose software system, such as a database management system, is classified as **dependent.** It is implemented as an application of, and consequently is dependent on, a database or DBMS to accomplish its functions [VA82]. The close connection between the two does not necessarily imply that the data dictionary is active with respect to the other software system [NA80], but most likely a data dictionary integrated with a DBMS will be an active facility, driving certain DBMS functions, aiding in the use of the DBMS, and enforcing the correct representation of the data by application programs using the data [MA83].

1.5 Summary

With the recognition of the importance of understanding a system's data as essential to the system's accurate development, as well as the increased awareness of data as a valuable corporate resource, more importance has been attached to tools to support the documentation, control, and communication of data-related information. The data dictionary is such a tool. Its usefulness extends across the life cycle of software development, into database administration, and into the arena of managing data as a corporate resource.

The data dictionary tool that I have developed and is described in the rest of this report supports the objectives of improved documentation, control, and communication of data definitions. As a research tool, it does not implement all of the facilities that would be required of a

commercial product, but it does provide a mechanism for capturing information about the data entities from an e-r-a requirements specification and for adding additional attribute information interactively. It is implemented as a stand-alone data dictionary with minimal active facilities. Facilities to enhance its active capabilities are planned, but are outside the scope of this implementation and report.

2. REQUIREMENTS

2.1 Overview

The data dictionary tool that is described in this report is part of a tool-set developed by a group of Kansas State University graduate students to begin on a prototype of a software development environment suitable for a fifth generation computer system. Since one of the objectives of software engineering for fifth generation computer systems is to lessen the burden of software generation [MO81], these tools attempt to address that goal by automating as much of the process as possible. An ideal scenario for the fifth generation environment would be one in which a computer processing procedure is directly synthesized from requirements specifications described in a natural language, and then generated and performed [MO81]. The focal input for the tools developed at Kansas State University is an e-r-a requirements specification describing entities, relationships, and attributes in a textual format. The syntax of the e-r-a specification is described in Appendix A.[2]

2.2 General Requirements

As a research tool, the data dictionary has a number of general requirements. It is important that it be flexible and easy to modify because the structure and content of the e-r-a requirements specification used as input may need to change. Although it is fairly

---

2. The e-r-a specification and syntax diagram were provided by Dr. David A. Gustafson, Department of Computer Science, Kansas State University.

certain that a requirements specification should contain information about entities and relationships in order to define a system [CH80], the exact syntax and use of keywords may need to be further defined or changed as experience dictates.

Along the same line of thought, the contents of the data dictionary itself should be flexible and fairly general. The data dictionary should accommodate, but not be restricted to, the syntax used in the e-r-a specification. For example, in the e-r-a specification, names of data items are enclosed in dollar signs; other sources of data items to be recorded in the data dictionary may not use that notation. The dictionary should provide for a set of attributes to describe the data entities and it should permit aliases (other names for the same data entity) to be defined. It should also support the software engineering principle of data abstraction [WA81] by allowing decomposition of data definitions. In other words, a data item should be allowed to be defined in terms of other data items or elements in order to group related items together. For example, a "flight reservation" entity might be defined as containing a number of other data items, such as passenger name, flight, origin, destination, date, and class of service. Some uses of the data may need to reference the individual date elements, whereas others may find a reference to the abstract data entity "flight reservation" more appropriate.

The data dictionary tool must be easy to use. It should provide a reasonable interaction with the user, and allow for some flexibility in entering definitions. A mechanism for extracting information from the data dictionary in the form of reports and a query capability to access a definition by name is also required. User documentation for the tool

must be provided.

It is outside the scope of this tool to provide consistency checking and validation of the dictionary's contents, although these are certainly important requirements for an effective data dictionary system.

The target implementation system for this tool is the UNIX™ operating system available at Kansas State University.

2.3 Specific Requirements

2.3.1 Input Requirements

One of the primary inputs to the data dictionary tool is a requirements document in the form of an e-r-a specification. An e-r-a specification contains information about entities (objects), relationships between those entities, and attributes. The format of the specification is specified in Appendix A. Essentially, it is a text file with keywords to identify the entities and their attributes. Each keyword is set apart from the textual description by a delimiter (a colon [:]). The syntax of each entry in the e-r-a specification is of the general form:

```
Id-Keyword : text
  attribute-keyword : text
  attribute-keyword : text
  attribute-keyword : text
```

Continuation lines essentially have a "null" attribute keyword, i.e., they begin with the delimiter (except for white space).

A data entity in the e-r-a specification is identified by one of the following keywords: "Input", "Output", "Input_output", "Type", "Data", and "Constant". The name of the data entity follows the keyword and

delimiter and is enclosed in special symbols (dollar signs [$]). Descriptive attributes, again identified by keywords, follow the identification line and are indented. The attribute keywords with informal meanings are shown in Figure 5, section 2.3.3.

The other source of information for the data dictionary is that supplied interactively by a user of the tool. This allows the user to record additional information about the data items beyond what is contained in the e-r-a specification. The interactive input is also text-oriented and supplied on a keyword basis. To provide maximum flexibility, the user can define his or her own keywords. No validation is required on the textual description supplied for the keyword attributes so as to permit the user to enter any description he or she likes.

2.3.2 Output Requirements

The primary output of the tool is a data dictionary file containing the data entries from the e-r-a specification combined with the additional information supplied by the user. The data dictionary must record attribute and relationship information about the data entities contained in the e-r-a specification. Each entity forms a separate definition in the data dictionary, identified by the "NAME" keyword and the name of the data entity. Attribute information will be explicitly recorded by keyword. Relationships will be recorded implicitly, unless the user chooses to define relationship keywords. The syntax of the data dictionary is formally defined in Appendix B.

For the sake of flexibility, the data dictionary will also be a keyworded text file with lines of text in the general form:

KEYWORD : description

where

KEYWORD	represents a data dictionary keyword.
:	is used as a delimiter.
description	is textual information describing that attribute in free-form format.

A blank keyword implies a continuation line; a blank line separates each definition.

The data dictionary produced by the tool can contain definitions of two kinds of data items: data elements and composite data items. A data element is the lowest level of definition and is not expected to be composed of other data entities for its definition; rather, it is defined in terms of possible values or range of values. A composite data item may be composed of other composite data items or data elements. The two classes of data entities and the prescribed set of attributes for each of them are shown in Figures 2 and 3. All of the prescribed attributes do not need to be used for each data item.

The other outputs of the data dictionary tool are reports of the data dictionary contents.

2.3.3 Functional Requirements

The data dictionary tool must perform essentially three functions. The first is to extract and record data-related information from the e-r-a specification. The second is to provide facilities for interactively completing the definitions. To support the objective of communicating

Data Element Keywords	
Keyword	*Meaning*
NAME	name of data element
DESCRIPTION	description
TYPE	type
RANGE	allowable range of values
VALUES	enumerated list of possible discrete values
UNITS	units of measure for the data element
SOURCE	source
DESTINATION	destination
ALIASES	other names for the same element

Figure 2. Data Element Keyword Template

Composite Data Item Keywords	
Keyword	*Meaning*
NAME	name of composite data item
DESCRIPTION	description
COMPOSITION	component data elements or other composite data items which comprise the definition
UNITS	units of measure used to describe the data item
ORGANIZATION	how the components relate to each other, access method, data element(s) keyed upon
SOURCE	source
DESTINATION	destination
ALIASES	other names for the same data item

Figure 3. Composite Data Item Keyword Template

the data definitions, the third function is to provide reporting and query capabilities. This functions are shown in the data flow diagram of Figure 4.

For the first function, the data dictionary must capture the data entities from the e-r-a specification along with as much descriptive

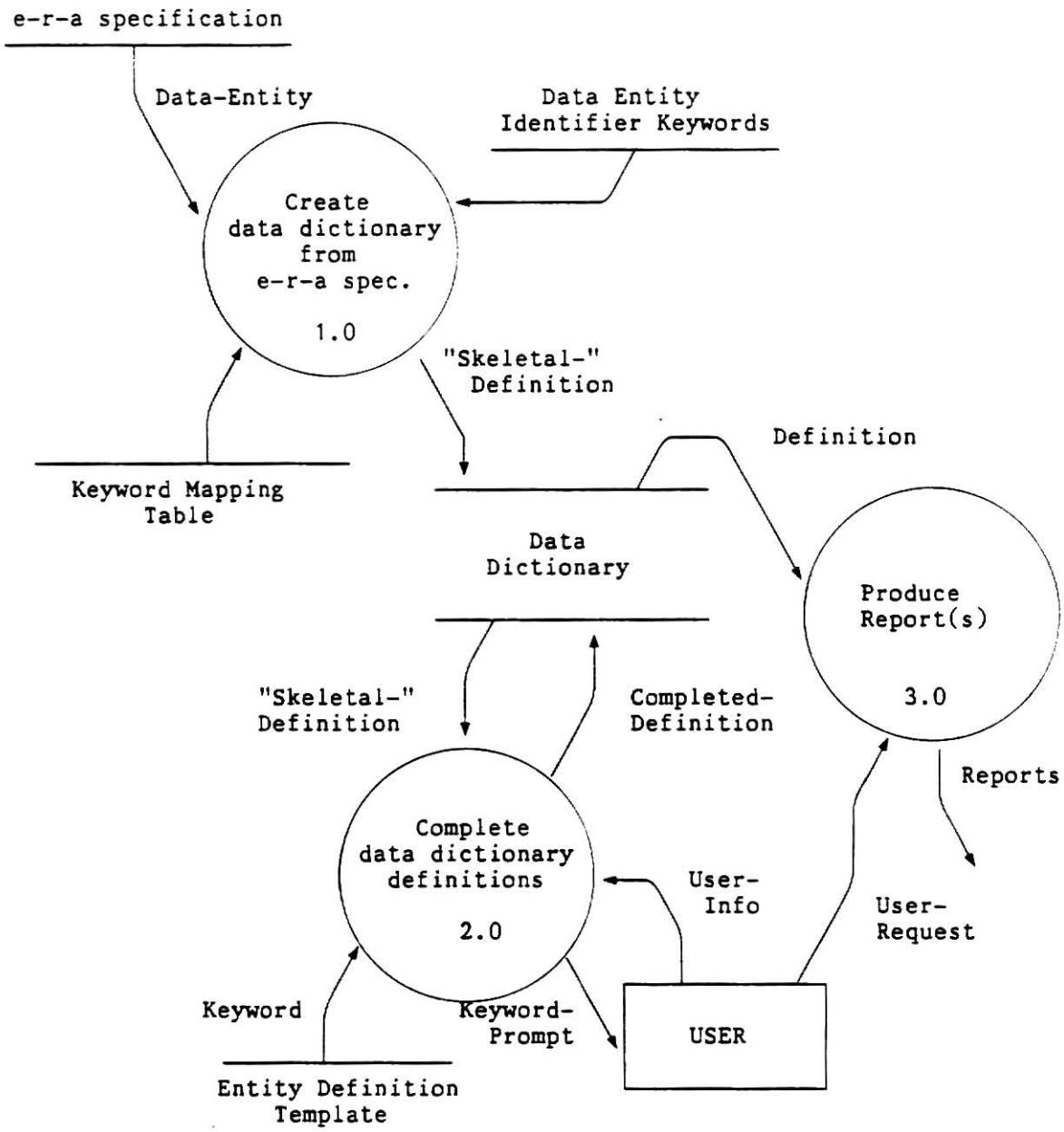

Figure 4. Data Flow Analysis of the Data Dictionary Tool

information about them as possible. The e-r-a data entity types are listed in section 2.3.1 and are also specified in Appendix A. The tool will read a file containing the e-r-a specification and extract only the data entities and their attributes to produce definition entries in the data dictionary. Each e-r-a data entity will correspond to a single definition. For each data entity, the name of the data entity (enclosed in dollar signs) will be used to identify the definition in the data dictionary and will be associated with the "NAME" keyword. The required mappings of the keywords following the identification line to the attribute keywords in the data dictionary are shown in Figure 5.

e-r-a Keyword	Meaning	DD Keyword
structure	Implies composition of other data items. Text of structure should contain data item(s) enclosed in "$"'s. e.g., $pieces$ ',' $position$	COMPOSITION
type	user-defined type (enclosed in "$"'s) or a predefined "keyword", such as character, integer, real, etc.	TYPE
enumeration	Possible discrete values.	VALUES
range	Range of permissible values.	RANGE
units	Unit of measurement for the data item.	UNITS
media	Used to indicate source or destination of external data items. Under "Input" or "Input/Output": Under "Output" or "Input/Output":	 SOURCE DESTINATION
comment	Text description of the data item's contents, usage, etc.	DESCRIPTION

Figure 5. Attribute Keyword Mappings

Because the requirements specification may not provide enough information about the data items, the second requirement for the tool is to provide interactive facilities to fill in other attribute information about the data items. An interactive facility will provide a question-and-answer dialogue style of prompting for information. The user will be allowed to enter new definitions or change existing ones (i.e., those created from the e-r-a specification).

With respect to adding a definition, the user will be prompted by a prescribed set of keywords according to the template specified (see section 2.3.2). He or she must be able to exit the keyword prompting mode at any time. The capability for specifying user-defined keywords is also a requirement.

An existing definition to be changed will be accessible by supplying the name of the item to be updated. The current definition will be displayed, and then the user will have the opportunity to update the contents of existing attributes or to add additional ones on a keyword basis. Again, user-defined keywords must be permitted.

The third functional requirement is to provide query and reporting capabilities for the dictionary's contents. This requirement is to support the role of a data dictionary as a communications tool by providing developers with some rudimentary capabilities for inspecting the completeness and accuracy of the data items specified. The required reporting capabilities are:

- a formatted listing of the data dictionary's contents

- a sorted listing according to defining term

- a "uses" listing which contains all definitions in which a specified data item is used

In addition, it must be possible to extract and display a single definition by specifying the name of the data item.

3. DESIGN

3.1 Design Strategy

The general approach taken in the design was to use as many existing capabilities available in the UNIX operating system as possible, and to develop any additional functions in a modular fashion. This strategy was taken in order to develop the prototype data dictionary tool as quickly as possible, given the time constraint of only a few weeks for its design and implementation. In addition, since text files were used for both the input e-r-a specification and the created data dictionary, the rich set of UNIX utility text filter programs lent themselves naturally to the task. By thinking in terms of the available tools, it was necessary only to solve the unique parts of the problem at hand, and interface to existing tools to do the rest. This is the same philosophy followed in the UNIX programming environment itself [KE76].

To support the requirements of flexibility and extensibility, I attempted to make the tool as table-driven as possible. Tables were used to record the data entity keywords used in the e-r-a specification and to hold the templates containing the data dictionary entity types. A table was also used to store the one-to-one mappings from the e-r-a keywords to those of the data dictionary. By limiting this information to tables, any keyword changes in either the e-r-a specification or in the data dictionary would require changes only to the tables rather than to the code itself. To further reduce the impact of keyword changes, the keywords were stored in files external to the programs and read in when needed.

The tool is partitioned into three major activities according to the

functional requirements discussed in the preceding chapter. These are shown in the high-level hierarchy diagram of Figure 6.

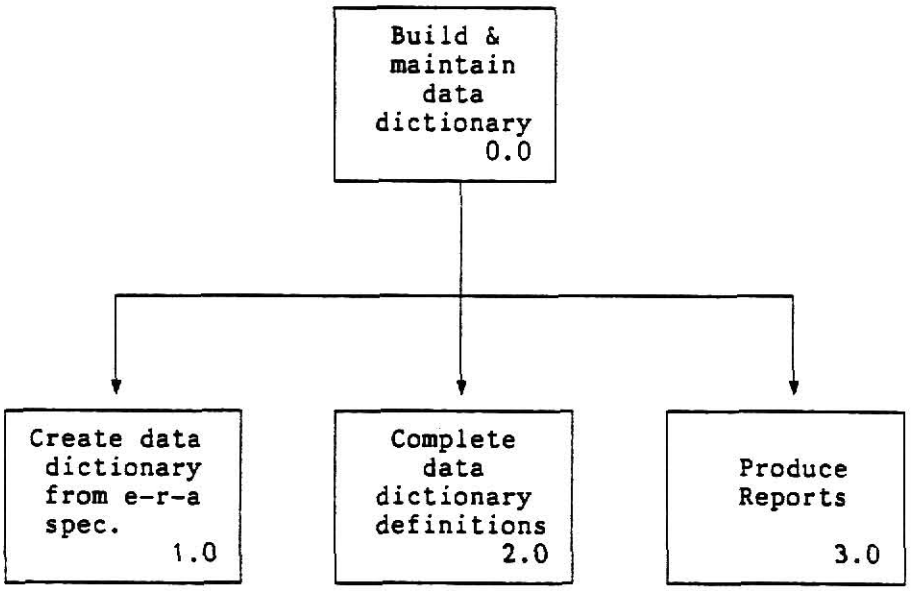

Figure 6. Hierarchy Diagram: Overview of the Data Dictionary Tool

3.2 Data Dictionary Creation

The first activity, which creates the data dictionary from the e-r-a specification, is composed of a set of transformation functions to map the e-r-a syntax to that of the data dictionary. The overall structure is shown in Figure 7. Since both the e-r-a specification and the generated data dictionary are keyworded text files, this activity lent itself to the use of some of the UNIX utility text filter tools such as *awk*, *grep*, and *sed*. The task was decomposed into a series of transformation steps. First, the data entities were extracted from the rest of the e-r-a specification based on the data identifier keywords stored in a table ("data.id.kws" file). The second step of mapping the

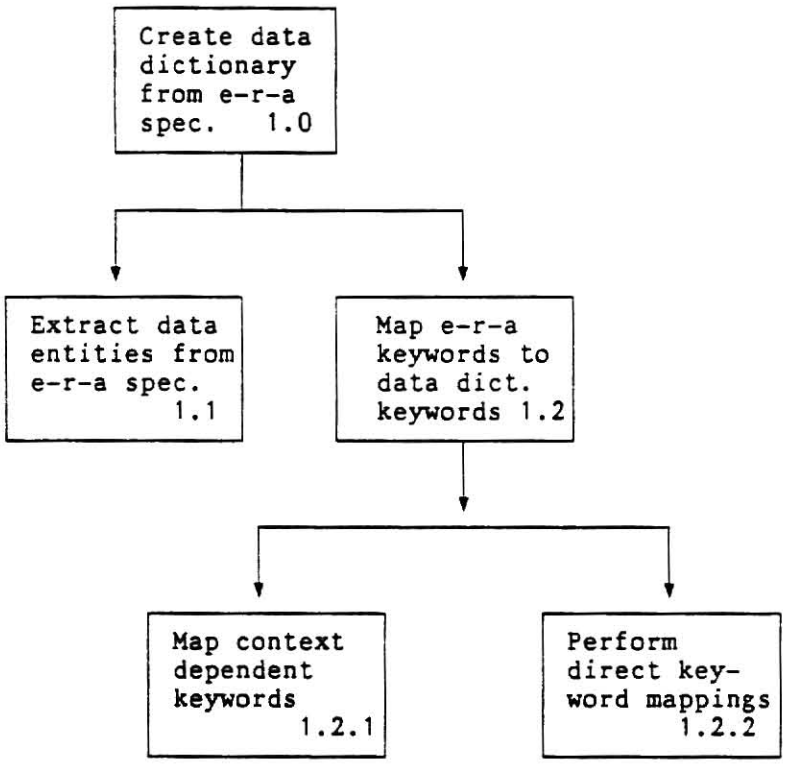

Figure 7. Hierarchy Diagram: Data Dictionary Creation

e-r-a keywords to those of the data dictionary was partitioned into first mapping any context-sensitive keywords and then mapping the other e-r-a keywords which have a one-to-one correspondence to the data dictionary keywords. The "media" keyword in the e-r-a specification is mapped to "SOURCE" in the data dictionary if it used to describe an "Input", to "DESTINATION" if it is used to describe an "Output", and to both if it is used in an "Input_output". This context-sensitive mapping required a separate pass through the e-r-a specification prior to mapping the other keywords. Following that transformation, all of the data identifier keywords are mapped to "NAME" and the other attribute keywords are mapped to their corresponding dictionary keywords as listed in Figure 5.

3.3 Interactive Update Capability

The second activity is the interactive capability for updating the data dictionary. The structure is shown in Figure 8.

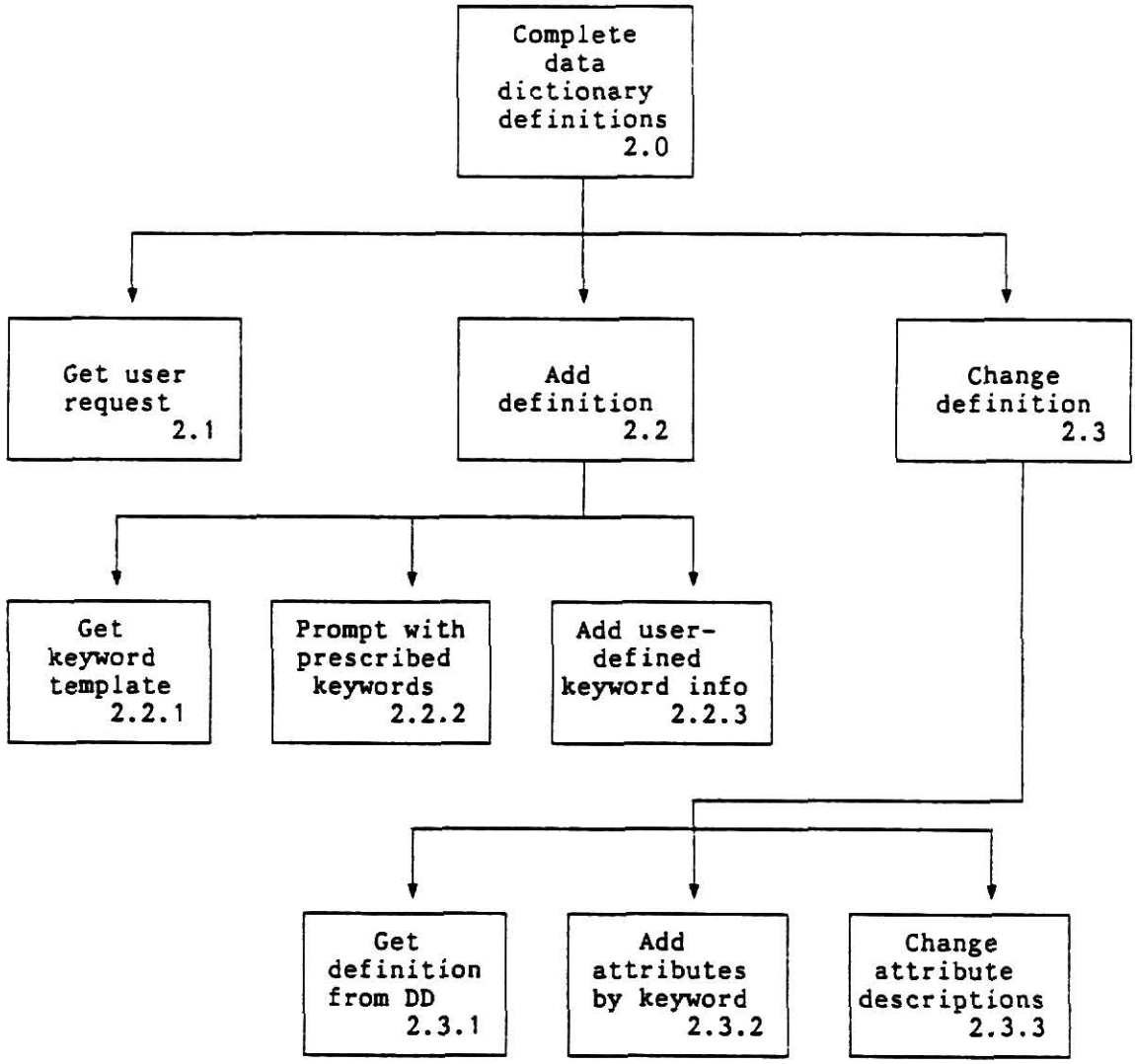

Figure 8. Hierarchy Diagram: Update Capability

The interactive capability is designed as a menu-driven function. After the menu is displayed, the user is prompted for the desired

operation and then control is passed to the appropriate function to handle the request. Four capabilities are provided: (1) to add a data element definition, (2) to add a composite data item definition, (3) to add a definition according to a user-defined template, and (4) to change an existing definition. The interactive capability was designed modularly in a combination of a UNIX shell script, awk utility programs, and C language modules. (The C programs to add a definition and to change a definition are in themselves stand-alone programs which can be directly invoked.)

All of the add-definition capabilities were designed to be handled by the same module as they each require only a different keyword template to be tailored to the specific entity type's keywords. The overall controlling script which prompts the user for the desired operation determines which keyword template is to be used and passes the template's file name as a parameter to the add-definition program. Each template is stored in an external file and read in during execution. User-defined keywords are permitted after the template keyword prompting. The user is allowed to enter as many definitions of the same entity type as desired before going on to a different add-definition operation or to the change operation.

A special character (+) on a line by itself is used to allow the user to exit the template keyword prompting prematurely. Continuation lines of text may be entered by preceding the carriage return with a backslash (\). A response of only a carriage return on the line allows the user to omit a prescribed keyword in the template from the actual definition being added without exiting the keyword prompting mode.

When the change operation is invoked, the script prompts for the name of the data entity to be changed. A separate function searches the data dictionary for that definition and, if found, passes it to the change-definition program. The change-definition program allows the user to add attributes by keywords or to change an existing attribute's description. Again, the special characters described above provide the user with some flexibility for controlling the session.

3.4 Report Generation

The third activity is actually composed of a set of commands to provide reporting and query capabilities. Each command drives the necessary routines to extract and format the requested information from the data dictionary. The specific commands and their use are listed in the User's Guide, Section 4.3, and are documented in Appendix D. Since this activity deals strictly with manipulating text files, it was designed to use UNIX text utility programs tailored to the specific syntax of the data dictionary.

4. IMPLEMENTATION

4.1 Implementation Details

The tool was developed on the UNIX System III operating system available at Kansas State University. Because a good deal of the tool was implemented in the UNIX shell and utility programs, it should be fairly portable to other UNIX systems running a compatible version of the UNIX operating system.

The data dictionary tool consists of 15 modules implemented with shell scripts and UNIX utility programs, such as *awk*, *grep*, and *sed*. Collectively, they contain about 430 lines of source code. The data dictionary creation function (*createdd*), and the query and reporting capabilities are written entirely in these tools. The controlling menu script for the interactive capability (*updatedd*) is written in the UNIX shell, and the definition extraction function for the update routine is handled by the shell and an *awk* program.

There are two modules coded in C language, consisting of about 750 lines of source code. These are the programs to add a definition (*adddef*) and to change a definition (*changedef*) invoked by the shell script *updatedd*. Listings of the source code may be found in Appendix E.

The requirements, design, and implementation of the tool was completed within a five-week period at Kansas State University and took approximately 50 hours to program and test.

4.2 Test Strategy

Because of the modular "building block" approach, each module was first tested individually to ensure the correct transformation of data from input to output. Each text filter program was tested and debugged, and the output examined to verify that the transformation expected was successfully performed. The interactive programs, *adddef* and *changedef*, were also tested individually with small test files before being combined with the shell script. The interactive programs were tested with both the data dictionary produced from the sample e-r-a specification and also with input entered in a different syntax to verify that the tool was not too restrictive. Each of the report generation commands were executed with the test data dictionary to verify the reporting capabilities.

After each function had been successfully tested, scripts combining the building blocks were tested. This was to ensure that the interfaces between the modules were consistent, and that the individual tools worked well together.

The interactive program was tested by an experimental user as well as the developer in order to get some objective feedback on the user prompts.

4.3 User's Guide

It is assumed that the user is familiar with the UNIX operating system and either has the data dictionary tool installed in the current directory or has included the pathname of its location in his or her $PATH environment variable.

The commands provided by the data dictionary tool are summarized below.

Command	Description
createdd erafile	Create data dictionary from e-r-a spec in "erafile".
updatedd ddfile	Invoke interactive capability using input file "ddfile".
printdd ddfile	Format and print data dictionary "ddfile".
sortdd ddfile	Sort data dictionary according to defining term.
sortprtdd ddfile	Sort and print data dictionary in one command.
getdef 'name' ddfile	Extract definition for 'name' from "ddfile".
uses 'name' ddfile	Print all definitions which use 'name'.

Figure 9. Data Dictionary Commands

A sample interaction with the data dictionary tool follows (commands are shown in boldface type).

Step 1. Create the data dictionary from a file called "ERA" containing a e-r-a specification as shown in Appendix A. The output is written to a file called "ddfile".

createdd ERA > ddfile

Step 2. Update the contents of the created data dictionary.

updatedd ddfile

The above command invokes the interactive procedure for updating a data dictionary file. In the following sample interaction, user responses are shown in boldface type.

```
                    Available Operations

                1       ADD data element definition
                2       ADD composite data item definition
                3       ADD definition--user-defined template
                4       CHANGE definition
                0       EXIT

What operation would you like?  1

You will be prompted to fill in information for these keywords:

        NAME
        DESCRIPTION
        TYPE
        RANGE
        VALUES
        UNITS
        SOURCE
        DESTINATION
        ALIASES

Enter   <CR>    to omit keyword from definition
         +      to exit keyword prompting
        \<CR>   to continue attribute description on next line

NAME : Item.A
DESCRIPTION : A is an item
TYPE : character
RANGE : <CR>
VALUES : A
UNITS : +

Do you wish to define other keywords for this definition?  y

KEYWORD : usage
USAGE : Used by Item B
KEYWORD : +

Do you want to add another definition?  n

What operation would you like?  4

Name of data item:  Item.A

--Current Definition--
Line    Keyword : Definition

1       NAME : Item.A
2       DESCRITPION : A is an item
3       TYPE : character
4       VALUES : A
5       USAGE : Used by Item B

Do you want to define any attributes?  y
```

```
        Available Escape Characters

Enter   <CR>    to omit keyword from definition
         +      to exit keyword prompting
        \<CR>   to continue attribute description on next line
         h      to display current definition

KEYWORD : security
SECURITY : none
KEYWORD : <CR>

--Current Definition--
Line    Keyword : Definition

1       NAME : Item.A
2       DESCRITPION : A is an item
3       TYPE : character
4       VALUES : A
5       USAGE : Used by Item B
6       SECURITY : none

Do you want to change any attribute descriptions?  y


        Available Escape Characters

Enter   <CR>    to omit keyword from definition
         +      to exit keyword prompting
        \<CR>   to continue attribute description on next line
         h      to display current definition

Enter unique KEYWORD or line number:  security

Current text: none

New text: Access restricted

Enter unique KEYWORD or line number:  +

--Current Definition--
Line    Keyword : Definition

1       NAME : Item.A
2       DESCRITPION : A is an item
3       TYPE : character
4       VALUES : A
5       USAGE : Used by Item B
6       SECURITY : Access restricted

Update for data item 'Item.A' completed.

What operation would you like?  0

updatedd completed
```

Step 3. Print a listing of the data dictionary.

printdd ddfile

Manual pages for the data dictionary commands are contained in Appendix D.

5. CONCLUSIONS

5.1 Evaluation of the Implementation

The data dictionary tool meets the specified requirements in a reasonable manner. It is flexible and can handle keyword changes with minimal impact. During the tool's development, the organization of the e-r-a specification changed as well as a number of the keywords it used. The data dictionary creation function that was already operational at the time of the specification changes required only fifteen minutes to modify to handle the revised keywords. The minimal amount of time required to make the changes can be attributed to the use of the UNIX text filter tools and to the fact that the tool is almost completely table-driven so that the knowledge of the keywords is hidden from the code itself.

The tool is fairly easy to use. The command line syntax is simple and patterned after the UNIX shell syntax. The commands are documented in the User's Guide, Appendix D. The interactive capability is workable, especially considering the short implementation interval, but it is not as sophisticated as a user might like. For example, the keyword template tables are currently updated via the UNIX text editor. A number of suggestions to improve this interaction are listed in the extensions below.

The use of the rich set of UNIX tools, such as *awk*, *grep*, and *sed*, greatly facilitated the rapid prototyping of the tool. The modules coded in combinations of those tools were developed and debugged in a fraction of the time as were the C language programs. A trade-off, however, is that they run significantly slower than their counterparts

in C would. Since efficiency was not an issue for this research tool, this drawback was not a problem. However, if the tool is used with large input files and/or speed does become an issue, some of the modules can be recoded in C language. The tool's overall structure already exists and has been tested which should minimize the effort required.

A problem we encountered with the independent tool-set philosophy used in the overall prototype project was the potential for inconsistent interfaces between some of the individual tools. How to combine the different tools to get an efficient instrumentation for a software project is a noted problem of a tool-set software development environment [HA82].

5.2 Extensions

For the data dictionary to serve as an effective knowledge base about data entities, a number of extensions are suggested:

- automatic consistency checking capabilities within the data dictionary to ensure that

    1. there are no multiply-defined terms.

    2. all composite data items and data elements used in a definition are in turn defined themselves (completeness).

    3. content of certain attributes is valid and "expected". This is important for other software using the data dictionary, such as module generators that require specific content in an attribute field rather than the freedom provided by this

tool.

- automatic consistency checking between the e-r-a specification and the data dictionary, i.e., all data items used in the specification are defined in the data dictionary.

- provide for recording of explicit relationships and linking of related items.

- provide for recording of process-related information, e.g., which programs or systems use the data items.

- enhanced active capabilities to provide automatic interfaces to other software tools such as module generators and programs to perform change impact analysis.

- automatic interfaces *from* other software components in order to update the contents of the data dictionary automatically.

- implement security and control measures to protect the contents of the data dictionary.

In addition to the above extensions to improve the power and usefulness of the data dictionary, a number of human factors suggestions are:

- to develop a forms-oriented approach for filling in or adding definitions on a CRT screen.

- to provide a more sophisticated facility for interactively updating the keyword templates.

- to develop a "layering" approach between the requirements definition tool and the data dictionary. In other words, develop

a mechanism between the requirements definition tool and the data dictionary to allow the user to "escape" the requirements definition tool to record some information about the data entity in the dictionary.

6. ACKNOWLEDGEMENTS

I would like to thank Dr. David A. Gustafson for serving as the Major Professor for this implementation project and report. I appreciate the time he has spent in discussions of the tool's capabilities and design alternatives, as well as the hours spent in reviewing and editing this report. Dr. Gustafson also provided the BNF syntax diagram for the e-r-a specification (Appendix A).

I would also like to thank the other members of my committee, Dr. Virgil E. Wallentine and Dr. William J. Hankley, for their interest in the project and reviewing this report within an extremely short time interval.

In addition, I appreciate the patient support and encouragement of my friends and colleagues.

The UNIX operating system has also been a valuable aid. In addition to providing the host environment for the project's implementation, this report was formatted and printed using the MM Memorandum Macros text formatting package.

REFERENCES

[BC77] British Computer Society, *Data Dictionary Systems Working Party Report*, London, 1977.

[CH80] Chen, P. P., ed., *Entity-Relationship Approach to Systems Analysis and Design*, Proceedings of the International Conference on Entity-Relationship Approach to Systems Analysis and Design, Los Angeles, Dec. 10-12, 1979, North-Holland, Amsterdam, 1980.

[CH83] Christian, K., *The UNIX Operating System*, John Wiley and Sons, New York, 1983.

[CR83] Cross, F. E., "The Module Generator: A Practical Definition for a Software Module", *Proceedings*, *COMPSAC 83*, IEEE Computer Society Press, 1983, pp 121-133.

[CU81] Curtice, R. M., and E. M. Dieckmann, "A Survey of Data Dictionaries", *DATAMATION*, March 1981, pp 135-158.

[DA80] Davis, A. M., "Automating the Requirements Phase: Benefits to Later Phases of the Software Life Cycle", *Proceedings*, *COMPSAC 80*, IEEE Computer Society Press, 1980, pp 42-48.

[DE78] De Marco, T., *Structured Analysis and System Specification*, Yourdon Inc., New York, 1978.

[DI81] Dickinson, B., *Developing Structured Systems*, Yourdon Press, New York, 1981.

[GA82] Galland, F. J., *Dictionary of Computing*, John Wiley and Sons, New York, 1982.

[GU79] *IPS Dictionary/Directory Concepts and Requirements*, GUIDE International Corp., Chicago, 1979.

[GU81] Gupta, P., U. Dayal, and A. G. Dale, "An Interactive Data Dictionary System to Support Logical Database Design", *Proceedings*, *COMPSAC 81*, IEEE Computer Society Press, 1981, pp 171-183.

[HA82] Hausen, H. L., and M. Mullerburg, "Software Engineering Environments: State of the Art, Problems and Perspectives",

Proceedings, *COMPSAC 82*, IEEE Computer Society Press, 1982, pp 326-333.

[KA83] Kahn, B. K., and E. W. Lumsden, "A User-Oriented Framework for Data Dictionary Systems", *DATA BASE*, Fall 1983, pp 28-36.

[KE76] Kernighan, B. W., and P. J. Plauger, *Software Tools*, Addison-Wesley, Reading, Mass., 1976.

[KE78] Kernighan, B. W., and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.

[KE84] Kernighan, B. W., and R. Pike, *The UNIX Programming Environment*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

[LE79] Leong-Hong, B., *et al.*, "Roles of the Data Dictionary System" (panel notes), *Proceedings*, *COMPSAC 79*, IEEE Computer Society Press, 1979, p 667.

[LI80] Lipka, S. E., "Some Issues in Requirements Definition", *Proceedings*, *COMPSAC 80*, IEEE Computer Society Press, 1980, pp 56-58.

[MA82a] Marca, D., and McGowan C., "Static and Dynamic Data Modeling for Information System Design", *6th International Conference on Software Engineering*, 1982, pp. 137-142.

[MA82b] Marselos, N. L., "DDFAC: An Automated Data Dictionary Facility", *Proceedings*, *1982 NSC Software Seminar*, Western Electric, 1982, pp 189-199.

[MA76] Martin, J., *Principles of Data-Base Management*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1976.

[MA83] Martin, J., *Managing the Data-Base Environment*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.

[MA80] Matsumoto, Y., and S. Kawakita, "A Method to Bridge Discontinuity Between Requirements Specification and Design", *Proceedings*, *COMPSAC 80*, IEEE Computer Society Press, 1980, pp 259-266.

[MO81] Moto-Oka, T., ed., *Fifth Generation Computer Systems*, *Proceedings of the International Conference on Fifth Generation*

Computer Systems, Tokyo, Japan, Oct. 19-22, 1981.

[MS80] *The Data Dictionary in Systems Development*, MSP, Inc., Lexington, Mass., 1980.

[NA80] National Bureau of Standards, "Guideline for Planning and Using a Data Dictionary System", *Federal Information Processing Standards Publication*, FIPS Pub 76, US Dept. of Commerce, August 20, 1980.

[NA83] National Bureau of Standards, "Specifications for a Federal Information Processing (FIPS) Data Dictionary System" (presentation slides), Institute for Computer Sciences and Technology, Washington, D. C., Nov. 17, 1983.

[RI80] Riddle, W. E., "Panel: Software Development Environments", *Proceedings*, *COMPSAC 80*, IEEE Computer Society Press, 1980, pp 220-224.

[RO77] Ross, D. T., and K. E. Schoman, Jr., "Structured Analysis for Requirements Definition", *IEEE Transactions on Software Engineering*, Vol. SE-3, Number 1, 1977, pp 6-15.

[SA83] Sastry, M.N., "Summary of Federal Information Processing Standards for Data Dictionary Systems", St. Paul, Minn., unpublished.

[SH83] Shapiro, E. Y., "Fifth Generation Project - a trip report", *Comm. ACM*, 26, Sept. 1983, pp 637-641.

[SI80] Sippl, C. J., and R. J. Sippl, *Computer Dictionary and Handbook*, 3rd ed., Howard W. Sams and Co., Inc., 1980.

[TE79] Teplitzky, P., "An Approach for Choosing a Programming Specification Methodology", *Proceedings*, *COMPSAC 79*, IEEE Computer Society Press, 1979, pp 128-135.

[VA82] Van Duyn, J., *Developing a Data Dictionary System*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[WA81] Wasserman, A. I., "Toward Integrated Software Development Environments", *Tutorial: Software Development Environments*, (originally presented at COMPSAC '81), IEEE Computer Society, 1981.

APPENDIX A E-R-A Specification Format

General Description

The era specification will consist of a set of frames. The order of the frame is not fixed. Each frame will contain information about one entity. Each frame will start on a newline. The first line in the frame will contain the keyword that describes the type of the entity and the name of the entity. The first letter in the type is capitalized. The type and the name are separated by a colon. At least one blank line will separate each frame.

The information in a frame is generally in the form of relations between this entity and other entities. Some of the information is in the form of attributes. An attribute gives information about this entity without referring to other entities. The order of these relations/attributes is not fixed.

Each relation/attribute is specified by a keyword that specifies the relation/attribute and its value. The value is either the name of the entity that has that relation or a text description of the attribute value. A colon separates the keyword and its value. Each relation/attribute starts on a new line. If a relation/attribute continues on to another line, the continuation line starts with a blank field followed by a colon. Multiple occurrences of a relation/attribute are represented by multiple occurrences of the keyword.

Entity Types

These entity types are not fixed. Additional entity types may be defined in the future. All entity types will start with a capital letter.

Activity
Type
Input
Output
Periodic_function
Input_output
Data
Constant
Comment
* additional entity types may be added at any time

Relations/Attributes

keywords
input
output
required_mode
necessary_condition
occurrence
assertion
action
comment
media
structure
type
enumeration
range
units
subpart_is
subpart_of
uses
* additional entity types may be added at any time

Syntax Description

```
<era_spec> ::=
        <era_title> <era_body> <mode_table>

<era_title> ::=
        PROCESS : <text>

<era_body> ::=
        <frame> | <frame> <era_body>

<frame> ::=
        <NL> <NL> <frame_header> <frame_body>
      | <NL> <NL> Comment : <text_lines>

<frame_header> ::=
        <i_o_data_header> : <i_o_data_name>
      | <function_header> : <CAPITAL_WORD>

<i_o_data_header> ::=
        Type | Input | Output | Input_output | Data
      | Constant | <CAPITAL_WORD>

<function_header> ::=
        Activity | Periodic_function | <CAPITAL_WORD>

<frame_body> ::=
        <relation> | <relation> <frame_body>

<relation> ::=
        <NL_B> <relation_type> : <relation_value>

<relation_type> ::=
        keywords | input | output | required_mode
```

```
      | necessary_condition | occurrence | assertion
      | action | comment | media | structure | type
      | enumeration | range | units
      | subpart_is | subpart_of | uses | <WORD>

<relation_value> ::=
        <text_lines> | <structure>

<structure> ::=
        <struct> | <struct> <NL_B> : <structure>

<struct> ::=
        <name> | <text> | <name> <structure> | <text> <structure>

<name> ::=
        <mode_name> | <i_o_data_name>

<i_o_data_name> ::=
        $ <WORD> $

<mode_name> ::=
        * <WORD> *

<mode_table> ::=
        <NL> <NL> MODE_TABLE <mode_list> <initial_mode>
         <transition_body>

<mode_list> ::=
        <mode> | <mode> <mode_list>

<mode> ::=
        <NL_B> Mode : <mode_name>

<initial_mode> ::=
        <NL> <NL_B> Initial_Mode : <mode_name>

<transition_body> ::=
        <NL> <NL_B> Allowed_Mode_Transitions : <transition_list>

<transition_list> ::=
        <transition> | <transition> <transition_list>

<transition> ::=
        <NL_B> <event> : <mode_name> -> <mode_name>

<event> ::=
        <i_o_data_name>
      | <i_o_data_name> = ' <text> '
      | <function_header>


<text_lines> ::=
        <text> | <text> <text_cont>

<text> ::=
        <WORD> | <WORD> <text>
```

```
<text_cont> ::=
        <NL_B> : <text> | <NL> : <text> <text_cont>

<NL> ::=
        '0 | '0 <NL>

<NL_B> ::=
        <NL> '    '
```

LEXICAL SCANNER INFORMATION

Tokens used in the productions above may begin with <char> or one of the following characters: *$,':-={}
Blanks can delimit tokens as well.

The following tokens are important above:

```
        <WORD>          ::= <char> | <char> <WORD>
        <CAPITAL_WORD>  ::= <capital_letter> <WORD>

<char> ::=
        <lower_case_char> | <symbol>

<lower_case_char> ::=
        a | b | ...| z | 0 | 1 | ... | 9

<symbol> ::=
        # | % | & | ( | ) | ? | _

<capital_letter> ::=
        A | B | ... | Z
```

There exists a set of "reserved word" tokens that includes:

```
        {keyboard,crt,internal,secondary_storage,NONE,every,mode}
```

SAMPLE E-R-A SPECIFICATION

```
PROCESS : Requirements specification for the chess program

Comment : as of 6/25/84 1:40 in ksu832:/usrb/we/era
        :
        : comment on specification:
        :    Not all of the activities necessary for this program
        : to be implemented are included in this description.
        : Some activities are not included if their activities
        : were determined by the other activities.  The activity
        : of interpreting the user's command was not included.

Type : $piece$
   structure : a string from the set {Kr,Kk,Kb,K,Q,Qb,Qk,Qr,p}

Type : $rank$
   structure : a string from the set {1,2,...8}

Type : $position$
   structure : $piece$ $rank$

Type : $piece_position$
   structure : $piece$ ',' $position$

Type : $board_matrix$
   structure : array[1..8,1..8] of $piece$ OR ' '

Input : $board_description$
   media     : keyboard
             : 'white' set of $piece_position$
             : 'black' set of $piece_position$
             : 'end'

Input : $name_of_game$
   media     : keyboard
   structure : 1 to 20 alphanumeric characters

Input : $new_user_input$
   media     : keyboard
   structure : any string

Input_output : $stored_board$
   media     : secondary_storage
   comment   : information to recreate the board configuration

Input_output : $chess_board$
   media     : internal
   structure : $board_matrix$

Comment : This page contains those Input entities which are
        : directly related to a command which the user of the
        : chess game might enter.  (As opposed to Input data
        : which is not a command, ie: $name_of_game$)
```

```
Input : $move$
   media     : keyboard
   structure : 'm' $position$ '-' $position$

Input : $display_board$
   media     : keyboard
   structure : 'display'

Input : $create$
   media     : keyboard
   structure : 'create'

Input : $concede$
   media     : keyboard
   structure : 'concede'

Input : $store$
   media     : keyboard
   structure : 'store' $name_of_game$

Input : $retrieve$
   media     : keyboard
   structure : 'retrieve' $name_of_game$

Comment : The remaining Input entities are 'pseudo commands'
        : intended to aid in manually exercising
        : Periodic_functions.  The entities were named by
        : switching the first and last words so as not to cause
        : name collisions with the Output entities.

Input : $mate_stale$
   media     : keyboard
   structure : 'stalemate'

Input : $limit_time$
   media     : keyboard
   structure : 'time_limit'

Input : $out_time$
   media     : keyboard
   structure : 'time_out'

Input : $check_input$
   media     : keyboard
   structure : 'input_check'

Comment : 1 Input entity above is unused.
        : 1 Input entity is omitted.

Output : $status$
   media     : crt
   structure : string from the set {'your move','check',
             : 'checkmate','concede'}

Output : $board_display$
   media     : crt
```

```
   structure : visually oriented display of current chess board

Output : $syntax_error$
   media     : crt
   structure : <cr> 'illegal, try again'

Output : $store_message$
   media     : crt
   structure : 'board stored'
   structure : 'storage failed'

Output : $retrieve_message$
   media     : crt
   structure : $name_of_game$ 'retrieved'
   structure : 'retrieval failed'

Output : $stalemate$
   media     : crt
   structure : 'stalemate occurred'

Output : $time_warning$
   media     : crt
   structure : 'this is a warning - 5 minutes elapsed'

Output : $time_out$
   media     : crt
   structure : 'too much time - game over'

Output : $move_message$
   media     : crt
   structure : <cr>
   structure : 'illegal move'

Output : $computer_move_message$
   media     : crt
   structure : 'computer moves from' $position$ 'to' $position$

Activity : Initialize_board
   keywords            : Standard_board,Initialize,Place_pieces
   input               : NONE
   output              : $chess_board$
   required_mode       : *START*
   necessary_condition : $start$
   assertion           : The output board is a correct representation
                       : of the standard starting configuration for
                       : chess

Activity : Create_special_board
   keywords            : Assign_positions,Place_pieces
   input               : $board_description$
   output              : $chess_board$
   required_mode       : *START*
   necessary_condition : $create$

Activity : Store_board
   keywords            : Store_game_status,Save_board
```

```
   input                : $name_of_game$
   input                : $chess_board$
   output               : $store_message$
   required_mode        : *NORMAL*
   necessary_condition  : $store$
   assertion            : the game is stored in file '$name_of_game$'

Activity : Retrieve_board
   keywords             : Retrieve_board
   input                : $name_of_game$
   output               : $chess_board$
   output               : $retrieve_message$
   required_mode        : *START*
   necessary_condition  : $retrieve$
   assertion            : Retrieves game stored in file '$name_of_game$'
                        : if successful

Comment : 1 Input item above is related to more than 1 other entity

Activity : Validate_user_move
   keywords             : Check_move,Check_m_status,Move_validation
   input                : $chess_board$
   input                : $move$
   output               : $move_message$
   required_mode        : *NORMAL*
   assertion            : If the move is illegal,
                        : the mode changes to *ILLEGAL*

Activity : Computer_Move
   comment              : used to be Move
   keywords             : Select_move,Select_status
   input                : $chess_board$
   output               : $chess_board$
   output               : $computer_move_message$
   output               : $status$
   required_mode        : *NORMAL*
   action               : mode may change to *END*
                        : if $status$ = 'checkmate' OR 'concede'

Activity : Update_board
   keywords             : Update_position,Update_status
   input                : $chess_board$
   input                : $move$
   output               : $chess_board$
   required_mode        : *NORMAL*

Activity : Display_board
   keywords             : Display
   input                : $chess_board$
   output               : $board_display$
   required_mode        : *NORMAL*
   required_mode        : *END*
   necessary_condition  : $display_board$

Comment : 1 Input item above is related to more than 1 other entity
```

```
Comment : for simplicity, pseudo Input entities exist to manually
        : exercise these Periodic_functions.

Periodic_function : Stalemate
   required_mode : *NORMAL*
   occurrence    : whenever a board configuration is repeated 3 times
   input         : $mate_stale$
   output        : $stalemate$
   action        : change mode to *END*

Periodic_function : Time_Limit
   required_mode : *NORMAL*
   occurrence    : whenever the user response time exceeds 5 minutes
   input         : $limit_time$
   output        : $time_warning$
   action        : NONE

Periodic_function : Time_Out
   required_mode : *NORMAL*
   occurrence    : whenever the user response time exceeds 10 minutes
   input         : $out_time$
   output        : $time_out$
   action        : change mode to *END*

Periodic_function : Input_Check
   required_mode : every mode
   occurrence    : whenever user input does not match allowed syntax
   input         : $check_input$
   output        : $syntax_error$
   action        : change mode to *ILLEGAL*


MODE_TABLE
   Mode : *ILLEGAL*
   Mode : *NORMAL*
   Mode : *START*
   Mode : *END*

   Initial_Mode : *START*

   Allowed_Mode_Transitions :
   $create$                              : *START* -> *NORMAL*
   $start$                               : *START* -> *NORMAL*
   $retrieve$                            : *START* -> *NORMAL*

   $status$ = 'checkmate'                : *NORMAL* -> *END*
   $status$ = 'concede'                  : *NORMAL* -> *END*
   $stalemate$                           : *NORMAL* -> *END*
   $time_out$                            : *NORMAL* -> *END*

   $move_message$ = 'illegal move'       : *NORMAL* -> *ILLEGAL*
   $syntax_error$                        : *NORMAL* -> *ILLEGAL*

   $syntax_error$                        : *START* -> *ILLEGAL*

   $syntax_error$                        : *END* -> *ILLEGAL*
```

```
   $new_user_input$                          : *ILLEGAL* -> *NORMAL*

   $'<cr>'$                                  : *END* -> *START*

Comment : 2 of the above transitions are unfirable.
        : 3 of the above transitions cause mode indeterminacy.
        : as specified here, *END* is not a terminal mode.
```

APPENDIX B Data Dictionary Format

DATA DICTIONARY SYNTAX SPECIFICATION

```
<data_dictionary> ::=
                <definition>
               | <definition> <data_dictionary>

<definition> ::=
                <definition_id> <definition_body> <blank_line>

<definition_id> ::=
                NAME <delimiter> <data_entity_name> <NL>

<definition_body> ::=
                <attribute_desc>
              | <attribute_desc> <definition_body>

<attribute_desc> ::=
                <attribute_keyword> <delimiter> <attribute_text>
              | <delimiter> <attribute_text>

<attribute_keyword> ::=
                DESCRIPTION | TYPE | RANGE | VALUES | SOURCE
              | DESTINATION | ALIASES | COMPOSITION | UNITS
              | ORGANIZATION | <user_defined_keyword>

<attribute_text> ::=
                <WORD> <NL>
              | <WORD> <attribute_text>

<data_entity_name> ::=
                <WORD>

<user_defined_keyword> ::=
                <UPPER_CASE_WORD>

<delimiter> ::=
                :

<blank_line> ::=
                <NL>
              | <NL> <blank_line>

<WORD> ::=
                <char>
              | <char> <WORD>

<UPPER_CASE_WORD> ::=
                <capital_letter>
              | <capital_letter> <UPPER_CASE_WORD>
```

```
<char> ::=
                <lower_case_char>
              | <capital_letter>
              | <symbol>
              | <digit>

<lower_case_char> ::=
                a | b | . . . | z

<capital_letter> ::=
                A | B | . . . | Z

<digit> ::=
                0 | 1 | . . . | 9

<symbol> ::=
                # | % | & | ( | ) | ? | _ | * | $ | , | .
              | ' | " | - | = | + | { | } | [ | ] | |
```

DATA DICTIONARY EXAMPLE
(As produced from the sample e-r-a specification)

```
NAME : $piece$
COMPOSITION : a string from the set {Kr,Kk,Kb,K,Q,Qb,Qk,Qr,p}

NAME : $rank$
COMPOSITION : a string from the set {1,2,...8}

NAME : $position$
COMPOSITION : $piece$ $rank$

NAME : $piece_position$
COMPOSITION : $piece$ ',' $position$

NAME : $board_matrix$
COMPOSITION : array[1..8,1..8] of $piece$ OR ' '

NAME : $board_description$
SOURCE : keyboard
COMPOSITION : 'white' set of $piece_position$
            : 'black' set of $piece_position$
            : 'end'

NAME : $name_of_game$
SOURCE : keyboard
COMPOSITION : 1 to 20 alphanumeric characters

NAME : $new_user_input$
SOURCE : keyboard
COMPOSITION : any string

NAME : $stored_board$
SOURCE : secondary_storage
DESTINATION : secondary_storage
DESCRIPTION : information to recreate the board configuration

NAME : $chess_board$
SOURCE : internal
DESTINATION : internal
COMPOSITION : $board_matrix$

NAME : $move$
SOURCE : keyboard
COMPOSITION : 'm' $position$ '-' $position$

NAME : $display_board$
SOURCE : keyboard
COMPOSITION : 'display'

NAME : $create$
SOURCE : keyboard
COMPOSITION : 'create'

NAME : $concede$
```

```
SOURCE : keyboard
COMPOSITION : 'concede'

NAME : $store$
SOURCE : keyboard
COMPOSITION : 'store' $name_of_game$

NAME : $retrieve$
SOURCE : keyboard
COMPOSITION : 'retrieve' $name_of_game$

NAME : $mate_stale$
SOURCE : keyboard
COMPOSITION : 'stalemate'

NAME : $limit_time$
SOURCE : keyboard
COMPOSITION : 'time_limit'

NAME : $out_time$
SOURCE : keyboard
COMPOSITION : 'time_out'

NAME : $check_input$
SOURCE : keyboard
COMPOSITION : 'input_check'

NAME : $status$
DESTINATION : crt
COMPOSITION : string from the set {'your move','check',
            : 'checkmate','concede'}

NAME : $board_display$
DESTINATION : crt
COMPOSITION : visually oriented display of current chess board

NAME : $syntax_error$
DESTINATION : crt
COMPOSITION : <cr> 'illegal, try again'

NAME : $store_message$
DESTINATION : crt
COMPOSITION : 'board stored'
COMPOSITION : 'storage failed'

NAME : $retrieve_message$
DESTINATION : crt
COMPOSITION : $name_of_game$ 'retrieved'
COMPOSITION : 'retrieval failed'

NAME : $stalemate$
DESTINATION : crt
COMPOSITION : 'stalemate occurred'

NAME : $time_warning$
DESTINATION : crt
```

```
COMPOSITION : 'this is a warning - 5 minutes elapsed'

NAME : $time_out$
DESTINATION : crt
COMPOSITION : 'too much time - game over'

NAME : $move_message$
DESTINATION : crt
COMPOSITION : <cr>
COMPOSITION : 'illegal move'

NAME : $computer_move_message$
DESTINATION : crt
COMPOSITION : 'computer moves from' $position$ 'to' $position$
```

APPENDIX C Sample Report

This appendix contains a sample output report from the data dictionary tool. The data dictionary created from the sample e-r-a specification contained in Appendix A was used as the input file for the command that generated this report.

Sorted E-R-A Data Entities

DATA DICTIONARY

Wed Jul 4 11:22:58 CDT 1984

```
NAME            : $board_description$
SOURCE          : keyboard
COMPOSITION     : 'white' set of $piece_position$
                : 'black' set of $piece_position$
                : 'end'

NAME            : $board_display$
DESTINATION     : crt
COMPOSITION     : visually oriented display of current chess board

NAME            : $board_matrix$
COMPOSITION     : array[1..8,1..8] of $piece$ OR ' '

NAME            : $check_input$
SOURCE          : keyboard
COMPOSITION     : 'input_check'

NAME            : $chess_board$
SOURCE          : internal
DESTINATION     : internal
COMPOSITION     : $board_matrix$

NAME            : $computer_move_message$
DESTINATION     : crt
COMPOSITION     : 'computer moves from' $position$ 'to' $position$

NAME            : $concede$
SOURCE          : keyboard
COMPOSITION     : 'concede'

NAME            : $create$
SOURCE          : keyboard
COMPOSITION     : 'create'

NAME            : $display_board$
SOURCE          : keyboard
COMPOSITION     : 'display'

NAME            : $limit_time$
SOURCE          : keyboard
COMPOSITION     : 'time_limit'

NAME            : $mate_stale$
SOURCE          : keyboard
COMPOSITION     : 'stalemate'

NAME            : $move$
SOURCE          : keyboard
COMPOSITION     : 'm' $position$ '-' $position$
```

```
NAME            : $move_message$
DESTINATION     : crt
COMPOSITION     : <cr>
COMPOSITION     : 'illegal move'

NAME            : $name_of_game$
SOURCE          : keyboard
COMPOSITION     : 1 to 20 alphanumeric characters

NAME            : $new_user_input$
SOURCE          : keyboard
COMPOSITION     : any string

NAME            : $out_time$
SOURCE          : keyboard
COMPOSITION     : 'time_out'

NAME            : $piece$
COMPOSITION     : a string from the set {Kr,Kk,Kb,K,Q,Qb,Qk,Qr,p}

NAME            : $piece_position$
COMPOSITION     : $piece$ ',' $position$

NAME            : $position$
COMPOSITION     : $piece$ $rank$

NAME            : $rank$
COMPOSITION     : a string from the set {1,2,...8}

NAME            : $retrieve$
SOURCE          : keyboard
COMPOSITION     : 'retrieve' $name_of_game$

NAME            : $retrieve_message$
DESTINATION     : crt
COMPOSITION     : $name_of_game$ 'retrieved'
COMPOSITION     : 'retrieval failed'

NAME            : $stalemate$
DESTINATION     : crt
COMPOSITION     : 'stalemate occurred'

NAME            : $status$
DESTINATION     : crt
COMPOSITION     : string from the set {'your move','check',
                : 'checkmate','concede'}

NAME            : $store$
SOURCE          : keyboard
COMPOSITION     : 'store' $name_of_game$

NAME            : $stored_board$
SOURCE          : secondary_storage
DESTINATION     : secondary_storage
DESCRIPTION     : information to recreate the board configuration
```

```
NAME            : $store_message$
DESTINATION     : crt
COMPOSITION     : 'board stored'
COMPOSITION     : 'storage failed'

NAME            : $syntax_error$
DESTINATION     : crt
COMPOSITION     : <cr> 'illegal, try again'

NAME            : $time_out$
DESTINATION     : crt
COMPOSITION     : 'too much time - game over'

NAME            : $time_warning$
DESTINATION     : crt
COMPOSITION     : 'this is a warning - 5 minutes elapsed'
```

APPENDIX D User's Guide

This appendix contains manual pages for the various data dictionary commands.

NAME

createdd - create data dictionary from e-r-a specification

SYNOPSIS

createdd filename

DESCRIPTION

createdd creates a data dictionary from an e-r-a specification. *filename* is expected to be a keyworded text file containing an e-r-a specification. createdd writes the created data dictionary on the standard output; therefore, it is advisable to direct the output to a file, i.e.,

```
createdd filename > outfilename
```

If no input file is given, or if the specified input file cannot be opened for reading, an error message will be printed on the standard error.

NAME

getdef - display single definition from data dictionary

SYNOPSIS

getdef 'name' filename

DESCRIPTION

getdef extracts a single definition according to the name of the defining term specified and displays it on the standard output. Single quotes are required around the term if it contains any special characters. In addition, any dollar signs ($) used in 'name' must ALSO be preceded by a backslash (\) to escape their special meaning to the shell and the program, e.g.,

getdef '\$name\$' filename

If an incorrect number of arguments is specified on the command line, or if the input file cannot be opened, an error message will be displayed.

NAME

printdd - format and print data dictionary

SYNOPSIS

printdd filename ["title"]

DESCRIPTION

printdd formats a data dictionary file for printing. *filename* is expected to contain the data dictionary to be printed; it may contain all or part of an actual data dictionary. "title" is optional; if specified, it will be used as the title of the report. Double quotes are required around the title if it contains embedded spaces. printdd writes the output on the standard output unless otherwise directed; therefore, the output of printdd is usually piped to a line printer command, i.e.,

```
printdd filename | lpr
```

or directed to a file for on-line perusal.

NAME

sortdd - sort data dictionary on defining term

SYNOPSIS

sortdd filename

DESCRIPTION

sortdd sorts the data dictionary in *filename* according to defining term. The output is sorted in ascending alphabetic order. The output is written to standard output, therefore, it is advisable to direct the output to a file, i.e.,

```
sortdd filename > outfile
```

If no input file is given, or if the specified input file cannot be opened for reading, an error message will be printed on the standard error.

NAME

sortprtdd - sort and format data dictionary for printing

SYNOPSIS

sortprtdd filename ["title"]

DESCRIPTION

sortprtdd combines the operations of sortdd and printdd into a single command for convenience. *filename* is expected to be a keyworded text file containing all or part of a data dictionary. "title" is optional; if specified, it will be used as the title of the listing. Double quotes are required around the title if it contains embedded spaces. The output is written to the standard output, unless directed otherwise. Hence, a common use of the command is

```
sortprtdd filename | lpr
```

SEE ALSO

printdd, sortdd

NAME

updatedd - update data dictionary interactively

SYNOPSIS

updatedd filename

DESCRIPTION

updatedd invokes an interactive question-and-answer facility to update a data dictionary file. *filename* is expected to be a keyworded text file containing the data dictionary to be updated. updatedd will display a menu of the available operations and prompt for the necessary input.

If no input file is specified, or if the input file cannot be opened, an error message will be printed on the standard error.

NAME

uses - get all definitions which use a specified term

SYNOPSIS

uses ´name´ filename

DESCRIPTION

uses extracts all definitions using the term specified by ´name´ from the data dictionary in *filename*. Single quotes are required around the term if it contains any special characters. In addition, any dollar signs ($) used in ´name´ must ALSO be preceded by a backslash (\) to escape their special meaning to the program, e.g.,

```
uses ´\$name\$´ filename
```

The output is written to the standard output; therefore, it is advisable to direct the output to a file, i.e.,

```
uses ´name´ filename > outfile
```

If an incorrect number of arguments is specified on the command line, or if the input file cannot be opened for reading, an error message will be printed on the standard error.

APPENDIX E Source Code Listings

Apr 22 08:23 1985 add.c Page 1

```
 1 #include <stdio.h>
 2 #include <ctype.h>
 3 #include "define.h"
 4
 5 FILE    *def_file;
 6 char    cont_char = '\\';
 7
 8 main (argc, argv)
 9 int     argc;
10 char    *argv[];
11 {
12
13         /*
14          * Add definition
15          *
16          * Usage: adddef kwfilename outfile
17          *
18          * Allows user to add definition(s) to data dictionary.
19          * Keyword template used is passed in 'kwfilename'.
20          * New definitions are written to 'outfile'.
21          */
22
23
24         extern  FILE    *def_file;      /* pointer to output file       */
25         extern  char    cont_char;      /* line continuation character  */
26
27         FILE    *fopen(), *kw_file;
28         char    *fgets();
29         char    kw[MAXKWLEN];           /* input keyword                */
30         char    text[MAXTEXT];          /* input text (attribute)       */
31         char    line[MAXTEXT];
32         char    kw_table[MAXNUMKW][MAXKWLEN];
33         char    *ptr, *question;
34         char    exit_char = '+';        /* escape-prompt character      */
35         char    *defn_id_kw = "NAME";   /* keyword that id's definition */
36         int     i, j;
37         int     indent;                 /* for continuation prompts     */
38         int     used_kws;               /* number of keywords in table  */
39         int     restart;                /* start-over flag              */
40         int     more;                   /* flag                         */
41         char    *strcpy(), *strchr();
42
43
44         if (argc != 3)
45         {
46                 fprintf(stderr, "\nUsage: adddef kwfilename outfile\n");
47                 exit(1);
48         }
49
50         if ((kw_file = fopen(*++argv, "r")) == NULL)
```

```
51          {
52                  fprintf(stderr, "\nCannot open template file '%s'.\n", *argv);
53                  exit(1);
54          }
55
56                  /*** initialize keyword array ***/
57
58          for (i=0; i <= MAXNUMKW; i++)
59                  for (j=0; j <= MAXKWLEN; j++)
60                          kw_table[i][j] = ' ';
61
62                  /*** load keyword table ***/
63
64          i = 0;
65          while (((ptr = fgets(kw, MAXKWLEN, kw_file)) != NULL)  &&
66            (i <= (MAXNUMKW - 1)))
67                  strcpy(kw_table[i++], kw);
68          fclose(kw_file);
69
70          if (i >= MAXNUMKW)
71          {
72                  fprintf(stderr, "\nError: too many keywords ");
73                  fprintf(stderr, "in template file '%s'\n", *argv);
74                  exit(1);
75          }
76          else
77                  used_kws = i-1;
78
79
80                  /*** take newlines off keywords in table ***/
81
82          i = 0;
83          while (i <= used_kws)
84          {
85                  if ((ptr = strchr(kw_table[i++], '\n')) != NULL)
86                          strcpy(ptr, "\0");
87                  else
88                          kw_table[i][MAXKWLEN-1] = '\0';
89          }
90
91                  /*** open output file ***/
92
93          def_file = fopen(*++argv, "w");
94
95
96                  /*** print instructions for user ***/
97
98          printf("\nYou will be prompted to fill in information ");
99          printf("for these keywords:\n");
100         for (i=0; i <= used_kws; i++)
```

```
101             printf("\t%s\n", kw_table[i]);
102         printf("\nEnter:\t<CR>\tto omit keyword from definition\n");
103         printf("\t %c\tto exit keyword prompting\n", exit_char);
104         printf("\t %c<CR>\tto continue attribute description on next line\n",
105                 cont_char);
106
107                 /*** add definitions until user is through ***/
108
109         more = YES;
110         while ( more == YES )
111         {
112                 /*** prompt for prescribed keywords ***/
113                 printf("\n");
114                 restart = NO;
115                 i = 0;
116                 while ( i <= used_kws )      /*** while kws in table ***/
117                 {
118                         /* prompt with keyword */
119                         printf("%s %c ", kw_table[i], DELIMITER );
120                         /* get attribute desc */
121                         fgets(text, MAXTEXT, stdin);
122                         if ((ptr = strchr(text, NL)) == NULL)
123                         {
124                                 text[MAXTEXT-2] = '\n';
125                                 text[MAXTEXT-1] = '\0';
126                                 printf("\n(Truncated)\n");
127                                 /* truncated, eat rest of line */
128                                 fgets(line, MAXTEXT, stdin);
129                         }
130                         if (text[0] == NL)
131                         {
132                                 /*** <CR> not allowed on id keyword ***/
133                             if (strncmp(defn_id_kw, kw_table[i], strlen(kw_table[i]))== 0)
134                             {
135                                 printf("\n*** %s is a required keyword ***\n",
136                                         kw_table[i]);
137                                 restart = YES;  /* set start-over flag */
138                                 break;
139                             }
140                             else
141                             {
142                                 i++;    /* point to next entry in table */
143                                 continue;
144                             }
145                         }
146                         if ((text[0] == exit_char) && (text[1] == NL))
147                                 break;       /* exit keyword prompting */
148                         fprintf(def_file, "%s %c ",
149                                 kw_table[i], DELIMITER );
150                         if (text[strlen(text) - 2] == cont_char)
```

```
151                             {
152                                     indent = strlen(kw_table[i]);
153                                     get_cont_lines(text, indent);
154                             }
155                             fprintf(def_file, "%s", text);
156                             i++;
157                     }
158
159                     if (restart == YES)
160                             /* start over again with first keyword */
161                             continue;
162
163                             /*** Allow for user-defined keywords ***/
164
165                     question = ("\nDo you wish to define other keywords for this definition?  ");
166
167                     if (get_answer(question) == YES)
168                     {
169                             while (1)
170                             {
171                                     printf("\nKEYWORD %c ", DELIMITER);
172                                     fgets(kw, MAXKWLEN, stdin);
173                                     if ((kw[0] == exit_char) && (kw[1] == NL))
174                                             break;
175                                     if ((ptr = strchr(kw, NL)) != NULL)
176                                             strcpy(ptr, "\0");
177                                     else
178                                     {
179                                             kw[MAXKWLEN-1] = '\0';
180                                             /* kw truncated, eat rest of line */
181                                             fgets(text, MAXTEXT, stdin);
182                                             printf("\n\n(Keyword truncated)\n");
183                                     }
184                                     for (i=0; i < strlen(kw); i++)
185                                             kw[i] = toupper(kw[i]);
186                                     if (strncmp(defn_id_kw, kw, strlen(kw)) == 0)
187                                     {
188                                         printf("\nCannot use %s for ", defn_id_kw);
189                                         printf("user-defined keyword.\n");
190                                         continue;
191                                     }
192                                     else
193                                             fprintf(def_file, "%s %c ", kw, DELIMITER);
194                                     printf("%s %c ", kw, DELIMITER);
195                                     fgets(text, MAXTEXT, stdin);
196                                     if ((ptr = strchr(text, NL)) == NULL)
197                                     {
198                                             text[MAXTEXT-2] = '\n';
199                                             text[MAXTEXT-1] = '\0';
200                                             fgets(line, MAXTEXT, stdin);
```

```
201                                     printf("\n(Truncated)\n");
202                             }
203                             if (text[strlen(text) - 2] == cont_char)
204                             {
205                                     indent = strlen(kw);
206                                     get_cont_lines(text, indent);
207                             }
208                             fprintf(def_file, "%s", text);
209                     }
210             }
211
212             fprintf(def_file, "\n");        /*  separate definitions by NL  */
213
214             question = ("\nDo you want to add another definition? ");
215             more = get_answer(question);
216         }   /*** end while (more == YES) ***/
217 }
```

```
218
219 get_cont_lines(text, indent)
220 char    text[];
221 int     indent;
222 {
223
224         /*
225          * Prompt and get continuation lines
226          * of attribute descriptions, filling
227          * in blanks for keywords of the continued
228          * lines.
229          */
230
231         extern  FILE    *def_file;
232         extern  char    cont_char;
233
234         int     continuation;   /* flag to indicate continuation */
235         int     j;
236         char    line[MAXTEXT], *ptr;
237
238         continuation = YES;
239
240         while (continuation == YES)
241         {
242                 text[strlen(text) - 2] = ' ';
243                 fprintf(def_file, "%s", text);
244                 for (j=0; j < indent; j++)
245                 {
246                         printf(" ");
247                         fprintf(def_file, " ");
248                 }
249                 printf(" %c ", DELIMITER);
250                 fprintf(def_file, " %c ", DELIMITER);
251                 fgets(text, MAXTEXT, stdin);
252                 if ((ptr = strchr(text, NL)) == NULL)
253                 {
254                         text[MAXTEXT-2] = '\n';
255                         text[MAXTEXT-1] = '\0';
256                         fgets(line, MAXTEXT, stdin);
257                         printf("\n(Truncated)\n");
258                 }
259                 if (text[strlen(text) - 2] != cont_char)
260                         continuation = NO;
261         }
262 }
```

```
263
264 get_answer(question)
265 char    *question;
266 {
267         char input[MAXTEXT];
268
269         while (1)
270         {
271                 printf("%s", question);
272                 fgets(input, MAXTEXT, stdin);
273                 switch(input[0])
274                 {
275                 case 'Y':
276                 case 'y':
277                         if (strncmp(input, "y", 1)   == 0 ||
278                             strncmp(input, "yes", 3) == 0 ||
279                             strncmp(input, "YES", 3) == 0)
280                                 return(YES);
281                         else
282                                 printf("Please answer 'yes' or 'no'.\n");
283                         break;
284                 case 'N':
285                 case 'n':
286                         if (strncmp(input, "n", 1)  == 0 ||
287                             strncmp(input, "no", 2) == 0 ||
288                             strncmp(input, "NO", 2) == 0)
289                                 return(NO);
290                         else
291                                 printf("Please answer 'yes' or 'no'.\n");
292                         break;
293                 default:
294                         printf("Please answer 'yes' or 'no'.\n");
295                 }
296         }
297 }
```

Apr 22 08:23 1985 align.delim Page 1

```
 1 BEGIN {
 2         FS = ":"          # field delimiter is :
 3         OFS = ":"         # for input and output
 4         TABSET = 8        # number of spaces in a "tab"
 5 }
 6 {
 7 if ((n = length($1)) < TABSET)
 8 {
 9         $1 = $1 "         " "      "
10         printf $0 "\n"
11 }
12 else if ((n = length($1)) < (TABSET + 7))
13 {
14         $1 = $1 "         "
15         printf $0 "\n"
16 }
17 else
18         printf $0 "\n"
19 }
20 END {
21 }
```

```
Apr 22 08:23 1985  attribute.kws Page 1

     1 comment/DESCRIPTION
     2 structure/COMPOSITION
     3 type/TYPE
     4 units/UNITS
     5 enumeration/VALUES
     6 range/RANGE
```

Apr 22 08:23 1985 change.c Page 1

```
 1 #include <stdio.h>
 2 #include <ctype.h>
 3 #include "define.h"
 4
 5 FILE    *out_file;
 6 char    cont_char = '\\';       /* continuation char for line   */
 7 char    exit_char = '+';        /* escape-prompting character   */
 8 char    help_char = 'h';        /* help character               */
 9 char    kw_table[MAXNUMKW][MAXKWLEN];
10 char    desc_tab[MAXNUMKW][MAXTEXT];
11
12 main(argc,argv)
13 int     argc;
14 char    *argv[];
15 {
16         /*
17          * Change definition
18          *
19          * Usage:  changedef defnfile outfile
20          *
21          * Allows user to change the definition passed
22          * in 'defnfile'.  Output (i.e., the changed
23          * definition is written to 'outfile'.
24          */
25
26         extern  char    kw_table[][MAXKWLEN];
27         extern  char    desc_tab[][MAXTEXT];
28         extern  char    cont_char;
29         extern  FILE    *out_file;
30
31         FILE    *fopen(), *def_file;
32         char    *fgets();
33         char    *ptr, line[MAXLINE];
34         char    *defn_id_kw = "NAME";   /* keyword to begin definition  */
35         char    *question;
36         int     i, j, line_no;
37         int     indent;         /* text indent for continuation lines   */
38         int     last_kw;        /* pointer to last kw in table          */
39         char    kw[MAXKWLEN];
40         char    text[MAXTEXT];
41         char    *strcpy(), *strchr(), *strncpy();
42
43
44         if (argc != 3)
45         {
46                 printf("\nUsage:  changedef defnfile outfile\n");
47                 exit(1);
48         }
49
50         if ((def_file = fopen(*++argv, "r")) == NULL)
```

```
51          {
52                  fprintf(stderr, "\nCannot open definition file '%s'\n", *argv);
53                  exit(1);
54          }
55
56
57                  /*** initialize arrays ***/
58
59          for (i=0; i <= MAXNUMKW; i++)
60                  for (j=0; j <= MAXKWLEN; j++)
61                          kw_table[i][j] = '\0';
62          for (i=0; i <= MAXNUMKW; i++)
63                  for (j=0; j<= MAXTEXT; j++)
64                          desc_tab[i][j] = '\0';
65
66
67                  /*** load tables ***/
68
69          last_kw = 0;
70          while (((ptr = fgets(line, MAXLINE, def_file)) != NULL) &&
71                 (last_kw <= (MAXNUMKW - 1)))
72          {
73                  i = 0;
74                  j = 0;
75                  if (line[i] == ' ')     /* continuation line */
76                  {
77                          while ((line[i] != DELIMITER) && (j < (MAXKWLEN -1)))
78                                  kw_table[last_kw][j++] = line[i++];
79                          --j;
80                          kw_table[last_kw][j] = '\0';
81                  }
82                  else                    /* line has a keyword */
83                  {
84                          while ( isalnum(line[i]) && (j < MAXKWLEN) )
85                                  kw_table[last_kw][j++] = line[i++];
86                          if (j >= MAXKWLEN)
87                          {
88                                  kw_table[last_kw][MAXKWLEN - 1] = '\0';
89                                  fprintf(stderr, "Keyword too long; truncated to %s\n",
90                                          kw_table[last_kw]);
91                          }
92                          else
93                                  kw_table[last_kw][j] = '\0';
94                          while ( line[i] == ' ')
95                                  i++;                    /*** skip blanks ***/
96                  }
97                  while ( line[i] == DELIMITER )
98                                  i++;                    /*** skip delimiter ***/
99                  while ( line[i] == ' ')
100                                 i++;                    /*** skip blanks ***/
```

```
101                 j = 0;
102                 while ((line[i] != '\n') && (line[i] != '\0')
103                         && (j < MAXTEXT))
104                         desc_tab[last_kw][j++] = line[i++];
105                 if (j >= MAXTEXT)
106                 {
107                         desc_tab[last_kw][MAXTEXT - 1] = '\0';
108                         fprintf(stderr, "\nDescription truncated %s\n",
109                                 desc_tab[last_kw]);
110                 }
111                 else
112                         desc_tab[last_kw][j] = '\0';
113                 last_kw++;
114         }
115
116         fclose(def_file);
117         --last_kw;      /*** incremented one too many in loop ***/
118         --last_kw;      /*** remove blank line from end of table ***/
119
120         /*** Display definition before changes ***/
121         display_def(last_kw);
122
123
124
125         /*** Add user-defined attribute(s) ***/
126
127         question = "\nDo you want to define any attributes?  ";
128         if (get_answer(question) == YES)
129         {
130                 print_help();
131                 while (1)
132                 {
133                         printf("\nKEYWORD %c ", DELIMITER);
134                         fgets(kw, MAXKWLEN, stdin);
135                         if (kw[0] == NL)
136                                 continue;       /* reprompt with KW */
137                         if ((kw[0] == exit_char) && (kw[1] == NL))
138                                 break;
139                         if ((kw[0] == help_char) && (kw[1] == NL))
140                         {
141                                 display_def(last_kw);
142                                 continue;
143                         }
144                         if ((ptr = strchr(kw, '\n')) != NULL)
145                                 strcpy(ptr, "\0");
146                         else
147                         {
148                                 kw[MAXKWLEN-1] = '\0';
149                                 /* kw truncated, eat rest of line */
150                                 fgets(text, MAXTEXT, stdin);
```

```
151                             printf("\n(Keyword truncated)\n");
152                     }
153                     for (i=0; i < strlen(kw); i++)
154                             kw[i] = toupper(kw[i]);
155                     if (strncmp(defn_id_kw, kw, strlen(kw)) == 0)
156                     {
157                             printf("\nCannot use '%s' for ", defn_id_kw);
158                             printf("user-defined keyword\n");
159                             continue;
160                     }
161                     last_kw++;
162                     if (last_kw >= MAXNUMKW)
163                     {
164                             fprintf(stderr,
165                                     "\nExceeded KW table size\n");
166                             exit(1);
167                     }
168                     printf("%s %c ", kw, DELIMITER);
169                             /* get attribute desc */
170                     fgets(text, MAXTEXT, stdin);
171                     if (text[0] == NL)
172                             continue;   /* no write, reprompt w/KW */
173                     if ((text[0] == exit_char) && (text[1] == NL))
174                             break;
175                     /* store keyword in next table entry */
176                     strncpy(kw_table[last_kw], kw, strlen(kw));
177                     if (text[strlen(text) - 2] == cont_char)
178                     {
179                             indent = strlen(kw);
180                             last_kw = get_cont_lines(text, indent, last_kw);
181                     }
182                     else
183                     {
184                             if ((ptr = strchr(text, '\n')) != NULL)
185                                     strcpy(ptr, "\0");
186                             else
187                             {
188                                     text[MAXTEXT-1] = '\0';
189                                     /* desc. truncated, eat rest of line */
190                                     fgets(line,MAXTEXT,stdin);
191                                     printf("\n\n(Truncated)\n");
192                             }
193                             strncpy(desc_tab[last_kw], text, strlen(text));
194                     }
195             }
196     }
197
198
199
200
```

```
201         /*** display current definition ***/
202         display_def(last_kw);
203
204
205         /*** Change attribute(s) ***/
206
207         question = ("\nDo you want to change any attribute descriptions?  ");
208         if (get_answer(question) == YES)
209         {
210                 print_help();
211                 while(1)
212                 {
213                         printf("\nEnter unique KEYWORD or line number:  ");
214                         fgets(kw, MAXKWLEN, stdin);
215                         if ((kw[0] == exit_char) && (kw[1] == NL))
216                                 break;
217                         if ((kw[0] == help_char) && (kw[1] == NL))
218                         {
219                                 display_def(last_kw);
220                                 continue;
221                         }
222                         if ((ptr = strchr(kw, '\n')) != NULL)
223                                 strcpy(ptr, "\0");
224                         else
225                         {
226                                 kw[MAXKWLEN-1] = '\0';
227                                 /* kw truncated, eat rest of line */
228                                 fgets(line, MAXTEXT, stdin);
229                                 printf("\n\n(Keyword truncated)\n");
230                         }
231                         j = 0;
232                         while ((j < strlen(kw)) && (isdigit(kw[j]) != 0))
233                                 j++;
234                         if (j >= strlen(kw))    /* input is a number */
235                         {
236                                 line_no = atoi(kw);
237                                 if ((line_no <= (last_kw + 1)) && (line_no > 0))
238                                         i = line_no - 1;
239                                 else
240                                 {
241                                         printf("\n\n\t*** Line number out of range ***");
242                                         display_def(last_kw);
243                                         continue;   /* go back & reprompt */
244                                 }
245                         }
246                         else
247                         {
248                                 for (i=0; i < strlen(kw); i++)
249                                         kw[i] = toupper(kw[i]);
250                                 i = 0;
```

```
251                                 while ((i <= last_kw) &&
252                                     (strncmp(kw_table[i], kw, strlen(kw)) != 0))
253                                         i++;
254                                 if ( i > last_kw)
255                                 {
256                                         printf("\nCan't find keyword %s\n", kw);
257                                         continue;
258                                 }
259                         }
260                         /* i points to table entry to be changed */
261                         printf("\nCurrent text: %s\n", desc_tab[i]);
262                         /* clear out old contents */
263                         for (j=0; j < MAXTEXT; j++)
264                                 desc_tab[i][j] = '\0';
265                         printf("\nNew text    : ");
266                         fgets(text, MAXTEXT, stdin);
267                         if ((ptr = strchr(text, '\n')) != NULL)
268                                 strcpy(ptr, "\0");
269                         else
270                         {
271                                 text[MAXTEXT-1] = '\0';
272                                 /* desc. truncated, eat rest of line */
273                                 fgets(line, MAXTEXT, stdin);
274                                 printf("\n\n(Truncated)\n");
275                         }
276                         strncpy(desc_tab[i], text, strlen(text));
277                 }
278         }
279
280         display_def(last_kw);
281
282
283                 /*** write table to outfile ***/
284         out_file = fopen(*++argv, "w");
285         write_tab(last_kw);
286         fclose(out_file);
287
288         printf("\nUpdate for data item '%s' completed.\n", desc_tab[0]);
289 }
```

```
290
291 print_help()
292 {
293
294         /*
295          * Print instructions for user
296          */
297
298         extern  char    exit_char;
299
300
301         printf("\n\tAvailable Escape Characters\n");
302         printf("\nEnter\t<CR>\tto not include keyword in definition\n");
303         printf("\t %c\tto exit keyword prompting\n", exit_char);
304         printf("\t%c\<CR>tto continue attribute description on next line\n",
305                 cont_char);
306         printf("\t %c\tto display current definition\n\n", help_char);
307
308 }
```

```
309
310 get_cont_lines(text, indent, last_kw)
311 char    text[];
312 int     indent;         /* text indent for con't lines  */
313 int     last_kw;        /* insert point in tables       */
314 {
315         /*
316          * Prompts for and gets continuation lines
317          * of text.
318          */
319
320         extern  char    kw_table[][MAXKWLEN];
321         extern  char    desc_tab[][MAXTEXT];
322         extern  char    cont_char;
323
324         int     continuation;
325         int     j;
326         char    *ptr, line[MAXTEXT];
327
328         continuation = YES;
329
330         while (continuation == YES)
331         {
332                 text[strlen(text) - 2] = ' ';  /* remove con't char */
333                 text[strlen(text) - 1] = '\0';  /* and NL           */
334                 strncpy(desc_tab[last_kw], text, strlen(text));
335                 last_kw++;
336                 if (last_kw >= MAXNUMKW)
337                 {
338                         fprintf(stderr, "\nExceeded KW table size\n");
339                         exit(1);
340                 }
341                 for (j=0; j < indent; j++)
342                 {
343                         printf(" ");
344                         kw_table[last_kw][j] = ' '; /* blank kw for con't */
345                 }
346                 printf(" %c ", DELIMITER);
347                 kw_table[last_kw][j] = '\0';
348                 fgets(text, MAXTEXT, stdin);
349                 if (text[strlen(text) - 2] != cont_char)
350                 {
351                         continuation = NO;
352                         if ((text[0] == NL) ||
353                                 ((text[0] == exit_char) && (text[1] == NL)))
354                         {
355                                 kw_table[last_kw][0] = '\0';
356                                 last_kw--;
357                         }
358                         else
```

Apr 22 08:23 1985 change.c Page 9

```
359                             {
360
361                                     if ((ptr = strchr(text, '\n')) != NULL)
362                                             strcpy(ptr, "\0");
363                                     else
364                                     {
365                                             text[MAXTEXT-1] = '\0';
366                                             /* desc. truncated, eat rest of line */
367                                             fgets(line, MAXTEXT, stdin);
368                                             printf("\n\n(Truncated)\n");
369                                     }
370                                     strncpy(desc_tab[last_kw], text, strlen(text));
371                             }
372                     }
373             }
374             return(last_kw);
375     }
```

Apr 22 08:23 1985 change.c Page 10

```
376
377 display_def(last_kw)
378 int     last_kw;        /* points to last entry in table */
379 {
380         /*
381          * Display current definition as stored
382          * in tables kw_table, desc_tab.
383          */
384
385         extern  char    kw_table[][MAXKWLEN];
386         extern  char    desc_tab[][MAXTEXT];
387
388         int     i;
389
390
391         printf("\n--Current Definition--\n");
392         printf("Line\tKeyword : Definition\n\n");
393         for (i=0; i <= last_kw; i++)
394                 printf("%d\t%s %c %s\n", i+1, kw_table[i],
395                         DELIMITER, desc_tab[i]);
396 }
397
```

```
398
399 get_answer(question)
400 char    *question;
401 {
402         char input[MAXTEXT];
403
404         while (1)
405         {
406                 printf("%s", question);
407                 fgets(input, MAXTEXT, stdin);
408                 switch(input[0])
409                 {
410                 case 'Y':
411                 case 'y':
412                         if (strncmp(input, "y", 1)   == 0 ||
413                             strncmp(input, "yes", 3) == 0 ||
414                             strncmp(input, "YES", 3) == 0)
415                                 return(YES);
416                         else
417                                 printf("Please answer 'yes' or 'no'.\n");
418                         break;
419                 case 'N':
420                 case 'n':
421                         if (strncmp(input, "n", 1)  == 0 ||
422                             strncmp(input, "no", 2) == 0 ||
423                             strncmp(input, "NO", 2) == 0)
424                                 return(NO);
425                         else
426                                 printf("Please answer 'yes' or 'no'.\n");
427                         break;
428                 default:
429                         printf("Please answer 'yes' or 'no'.\n");
430                 }
431         }
432 }
```

```
433
434 write_tab(last_kw)
435 int     last_kw;
436 {
437         /*
438          * Write out contents of tables (i.e.,
439          * the changed definition) to outfile.
440          */
441
442         extern  char    kw_table[][MAXKWLEN];
443         extern  char    desc_tab[][MAXTEXT];
444         extern  FILE    *out_file;
445
446         int     i;
447
448
449
450
451         for (i=0; i <= last_kw; i++)
452                 fprintf(out_file, "%s %c %s\n", kw_table[i],
453                         DELIMITER, desc_tab[i]);
454
455         fprintf(out_file, "\n");    /*** blank line between defns ***/
456
457 }
```

```
Apr 22 08:23 1985  comp.data.kws Page 1

     1 NAME
     2 DESCRIPTION
     3 COMPOSITION
     4 UNITS
     5 SOURCE
     6 DESTINATION
     7 ALIASES
```

Apr 22 08:23 1985 createdd Page 1

```
 1 :
 2 : createdd: create data dictionary from e-r-a spec
 3 :
 4 : usage: createdd era-filename
 5 :
 6 : Data entities are extracted from the specified
 7 : e-r-a file based on the data entity type keywords
 8 : contained in the file 'data.id.kws'.
 9 : Data entity types are mapped to the data dictionary
10 : keyword NAME.  The "media" attribute is mapped
11 : to the data dictionary keywords SOURCE or DESTINATION,
12 : depending on context.  All other attribute mappings
13 : are contained in the file 'attribute.kws'.
14
15 :    check for existence and readability of file on command line
16 case $# in
17 0)      echo 'Usage: createdd era-filename' 1>&2 ; exit
18 esac
19
20 if test -r $1
21 then
22 :       /** get data entities from e-r-a spec **/
23        get.data.ent $1 > /tmp/bhh$$data
24
25 :       /** map "media" keyword to dd keywords **/
26        awk -f mapmedia /tmp/bhh$$data > /tmp/bhh$$med
27
28 :       /** add required syntax to keyword files for "sed" **/
29        sed 's/$/[ ]*:/
30             s.$./NAME :/.
31             s.^.s/.'  data.id.kws > tmp.mapname
32
33        sed 's./.[ ]*:/.
34             s.$. :/.
35             s.^.s/[ ]*.' attribute.kws > tmp.mapattributes
36
37 :       /** map other attribute keywords **/
38        sed -f tmp.mapname /tmp/bhh$$med |
39        sed -f tmp.mapattributes
40
41 :       /** cleanup temporary file **/
42        rm tmp.*
43
44 else
45        echo 'Cannot open input file' $1 1>&2
46        exit
47 fi
```

```
Apr 22 08:23 1985  data.el.kws Page 1

    1 NAME
    2 DESCRIPTION
    3 TYPE
    4 RANGE
    5 VALUES
    6 UNITS
    7 SOURCE
    8 DESTINATION
    9 ALIASES
```

Apr 22 08:23 1985 data.id.kws Page 1

```
1 Input
2 Output
3 Input_output
4 Type
5 Data
6 Constant
```

Apr 22 08:23 1985 define.h Page 1

```
1 #define MAXKWLEN          15      /* max length of keyword */
2 #define MAXNUMKW          20      /* max number of keywords */
3 #define MAXTEXT           55      /* max length of text string accepted */
4 #define MAXLINE           80
5 #define YES               1
6 #define NO                0
7 #define NL                '\n'
8 #define DELIMITER         ':'     /* delimiter between keyword and description */
```

Apr 22 08:23 1985 edit.for.prt Page 1

```
 1 cat <<!
 2 .ll 75
 3 .sp 4
 4 .ce
 5 $2
 6 .sp 2
 7 .ce 3
 8 DATA DICTIONARY
 9 .sp 1
10 !
11 date
12 cat <<!
13 .ce 0
14 .nf
15 .sp 1
16 .in +10
17 !
18 cat <<!
19 .de HD
20 'sp 4
21 .ns
22 ..
23 .de FT
24 'bp
25 ..
26 .wh 0 HD
27 .wh -7 FT
28 !
29
30 :  line up colons in data dictionary file
31
32 awk -f align.delim $1
```

Apr 22 08:23 1985 get.data.ent Page 1

```
 1 :     This shell script is designed to
 2 :     extract data entities from e-r-a spec
 3 :     according to data entity-identifying
 4 :     keywords in the file "data.id.kws".
 5
 6 case $# in
 7 0)        echo 'Usage: get.data.ent erafilename' 1>&2; exit
 8 esac
 9
10 :     combine e-r-a entries into single text line
11 sed 's/^[        ]*$/%/' $1 | awk -f join.awk > /tmp/bhh$$a
12
13 :     add proper syntax to data entity type keywords
14 :     in file 'data.id.kws' for "egrep"
15 sed 's/$/[ ]*:/' data.id.kws > tmp.kws
16
17 :     extract data entities by keyword
18 egrep -f tmp.kws /tmp/bhh$$a > /tmp/bhh$$b
19
20 :     reformat back to multiple lines
21 awk -f split.awk /tmp/bhh$$b
22
23 :     cleanup temporary files
24 rm tmp.kws
```

Apr 22 08:23 1985 getdef Page 1

```
 1
 2 :
 3 :  usage: getdef 'name' ddfilename
 4 :
 5 :
 6 :  check command line usage
 7 case $# in
 8 0|1)    echo "Usage:  getdef 'name' ddfilename" 1>&2; exit;;
 9 2)      ;;
10 *)      echo "Usage:  getdef 'name' ddfilename" 1>&2; exit;;
11 esac
12
13 :  check that ddfile is readable
14 if test ! -r $2
15 then
16         echo 'Cannot open input file' $2 1>&2
17         exit
18 fi
19
20 cat $2 | getdef.awk "$1"
```

Apr 22 08:23 1985 getdef.awk Page 1

```
awk '
BEGIN {
        dataname = "'$1'"       # name of data item to be found
        defnid = "NAME"         # "NAME" keyword ids beginning of definition
        found = "NO"
}

{
if ($1 == defnid && $3 == dataname)
{
        found = "YES"
        print
        rc = getline
        if (rc != 0)            # if not end of file
                while ($1 != "NAME" && rc != 0)
                {
                        print
                        rc = getline
                }
}

}
END {
        if (found == "NO")
                printf("\nData item %s not found\n", dataname)
}'
```

Apr 22 08:23 1985 join.awk Page 1

```
 1 #
 2 #  awk file to join separate lines (of an e-r-a spec entity)
 3 #  into single text line with the former lines separated by "#"
 4 #
 5 {
 6 if ($1 != "%")
 7         printf "%s#", $0
 8 else
 9         printf "\n"
10 }
```

Apr 22 08:23 1985 mapmedia Page 1

```
 1 #
 2 # awk file to map "media" keyword from e-r-a spec
 3 # to SOURCE and/or DESTINATION keyword depending on context
 4 #
 5 BEGIN {
 6         mf = 3          # initialize media flag
 7 }
 8
 9 $1 == "Input"    { mf = 0 }
10 $1 == "Output"   { mf = 1 }
11 $1 == "Input_output"     { mf = 2 }
12 $1 == "media" {
13         if ( mf == 0 )
14         {
15                 $1 = "SOURCE"
16                 print
17         }
18         else if ( mf == 1 )
19         {
20                 $1 = "DESTINATION"
21                 print
22         }
23         else if ( mf == 2 )
24         {
25                 $1 = "SOURCE"
26                 print
27                 $1 = "DESTINATION"
28                 print
29         }
30         else
31         {
32                 printf "%s keyword with unexpected data entity\n", $1 > "errors"
33         }
34         mf = 3          # reset media flag
35         next            # skip to next line of input
36 }
37 { print }
```

Apr 22 08:23 1985 printdd Page 1

```
 1
 2 :  Usage:  printdd ddfilename "title"
 3 :
 4 :  Formats and prints data dictionary in "ddfilename"
 5 :  to stdout.  "title" is optional and contains text
 6 :  to be used as the title of the listing.
 7 :
 8
 9 case $# in
10 1)       ;;
11 2)       ;;
12 *)       echo 'Usage:  printdd ddfilename ["title"]' 1>&2; exit;;
13 esac
14
15 if test ! -r $1
16 then
17          echo 'Cannot open input file' $1 1>&2
18          exit
19 fi
20
21 edit.for.prt $1 "$2" |
22 nroff
```

Apr 22 08:23 1985 sortdd Page 1

```
 1
 2 :
 3 :  Usage:  sortdd ddfilename
 4 :
 5 :  Sorts input data dictionary "ddfilename" in alphabetic
 6 :  order on defining term.  Output is on stdout.
 7 :
 8
 9 :  check command line arguments
10 case $# in
11 1)       ;;
12 *)       echo 'Usage:  sortdd ddfilename' 1>&2; exit
13 esac
14
15 if test ! -r $1
16 then
17          echo 'Cannot open input file' $1 1>&2
18          exit
19 fi
20
21 sed 's/^[        ]*$/%/' $1 | awk -f join.awk |
22 sort -f |
23 awk -f split.awk
```

Apr 22 08:23 1985 sortprtdd Page 1

```
 1
 2 :
 3 :  usage:  sortprtdd ddfilename ["title"]
 4 :
 5 :  Sorts and prints data dictionary in "ddfilename"
 6 :  with one command.  "title" is optional, and
 7 :  if specified, is used as the title of the listing.
 8 :
 9
10 :  check command line syntax
11 case $# in
12 1)       ;;
13 2)       ;;
14 *)       echo 'Usage:  sortprtdd ddfilename ["title"]' 1>&2; exit
15 esac
16
17 if test ! -r $1
18 then
19          echo 'Cannot open input file' $1 1>&2
20          exit
21 fi
22
23 sortdd $1 > /tmp/bhh$$srt
24 printdd /tmp/bhh$$srt "$2"
```

Apr 22 08:23 1985 split.awk Page 1

```
 1 #
 2 # awk file to put fields separated by FS
 3 # on separate lines
 4 #
 5 BEGIN {
 6         FS = "#"        # input field separator
 7 }
 8
 9 NF > 0 {
10         for (i=1; i<= NF; i++)
11                 printf "%s\n", $i
12 }
```

Apr 22 08:23 1985 splitddout.awk Page 1

```
awk '
BEGIN {
        dataname = "'$1'"       # name of data item to be found
        defnid = "NAME"         # "NAME" keyword ids beginning of definition
        found = "NO"
}

{
if ($1 == defnid && $3 == dataname)
{
        found = "YES"
        print >>"BHHchgfile"
        rc = getline
        if (rc != 0)            # if not end of file
        {
                while ($1 != "NAME" && rc != 0)
                {
                        print >>"BHHchgfile"
                        rc = getline
                }
                if (rc != 0)
                        print >>"BHHfile2"
        }
}
else
{
        if (found == "NO")
                print >>"BHHfile1"
        else
                print >>"BHHfile2"
}
}

END {
        if (found == "NO")
                printf("\n%s not found\n", dataname)
}'
```

Apr 22 08:23 1985 updatedd Page 1

```
 1 :
 2 : usage: updatedd dd-filename
 3 :
 4 : trap handling
 5 :
 6 : check for correct command line usage
 7 case  $# in
 8 0)      echo 'Usage:  updatedd ddfilename' 1>&2; exit;;
 9 1)      ;;
10 *)      echo 'Usage:  updatedd ddfilename' 1>&2; exit;;
11 esac
12 :
13 : check that data dictionary file can be opened
14 if test ! -w $1
15 then
16         echo 'Cannot open input file' $1 1>&2
17         exit
18 fi
19 :
20 cat >echomenu <<!
21 echo '\n\n\t\t\tAvailable Operations\n'
22 echo '\t\t1\tADD data element definition'
23 echo '\t\t2\tADD composite data item definition'
24 echo '\t\t3\tADD definition--user-defined template'
25 echo '\t\t4\tCHANGE definition'
26 echo '\t\t0\tEXIT'
27 !
28 chmod 700 echomenu
29 echomenu
30 :
31 ddfile=$1
32 :
33 while :
34 do
35         echo '\nWhat operation would you like?  \c'
36         read op
37         case $op in
38         1)  adddef data.el.kws BHHtmp;
39             cat -s BHHtmp >>$ddfile;
40             rm -f BHHtmp;;
41
42         2)  adddef comp.data.kws BHHtmp;
43             cat -s BHHtmp >>$ddfile;
44             rm -f BHHtmp;;
45
46         3)  echo '\nWhat is the name of your keyword template file?  \c'
47             read kwfile;
48             if test -r $kwfile
49             then
50                 adddef $kwfile BHHtmp
```

```
51                  cat -s BHHtmp >>$ddfile
52                  rm -f BHHtmp
53              else
54                  echo 'Cannot open template file' $kwfile 1>&2
55              fi;;
56
57          4)  echo 'Name of data item:  \c';
58              read dataname;
59              cat $ddfile | splitddout.awk $dataname;
60              if test -f BHHchgfile
61              then
62                  changedef BHHchgfile BHHtmp
63                  cat -s BHHfile1 BHHtmp BHHfile2 > $ddfile
64                  rm -f BHHfile1 BHHchgfile BHHfile2 BHHtmp
65              fi;;
66
67          0)  echo 'updatedd completed';
68              rm echomenu;
69              exit;;
70
71          h|H|*)  echo 'Please enter one of the following menu numbers:\n';
72                  echomenu;;
73          esac
74  done
```

Apr 22 08:23 1985 uses Page 1

```
 1 :
 2 :  usage:  uses 'name' ddfilename
 3 :
 4 :  Extracts all definitions from data dictionary
 5 :  in "ddfilename" that contain the term 'name'.
 6 :  The single quotes are required around 'name' if
 7 :  it contains special characters. In addition,
 8 :  any "$" used in name must ALSO be preceded by a
 9 :  backslash.
10 :
11
12 :  check command line syntax
13 case $# in
14 1)       echo "Usage:  uses 'name' ddfilename" 1>&2; exit;;
15 2)       ;;
16 *)       echo "Usage:  uses 'name' ddfilename" 1>&2; exit;;
17 esac
18
19 if test ! -r $2
20 then
21         echo 'Cannot open input file' $2 1>&2
22         exit
23 fi
24
25 sed 's/^[       ]*$/%/' $2 | awk -f join.awk |
26 egrep "$1" |
27 awk -f split.awk
```

CREATING A DATA DICTIONARY FROM A REQUIREMENTS SPECIFICATION

by

BETH HUHN HOFFMAN

B. A., Bradley University, 1978

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

The view of what is important for developing good software has evolved over the last two decades from a procedural-oriented view to one that is concerned with understanding data. With this recognition of the significance of understanding a system's data as essential to the system's accurate development, more importance has been attached to data-oriented methodologies and to tools to support them. A data dictionary is a repository of data about data. It contains the name of each data item, its definition, and perhaps information about its origin and usage. The term is also generally understood to include the procedures necessary to build and maintain the contents of the data dictionary.

The data dictionary tool which is described in this report supports the objectives of improved documentation, control, and communication of a system's data definitions. It provides a mechanism for capturing information about the data entities from a requirements specification which uses an entity-relationship-attribute (e-r-a) approach. It also provides facilities for completing the definitions and for reporting information from the created data dictionary.

The tool was implemented on the UNIX™ operating system available at Kansas State University. It was written in a combination of UNIX shell programs, text filters, and C language programs. The use of the rich set of UNIX tools greatly facilitated the quick development of this prototype tool.